Mauro Perotti Gabriella Cangemi

Impariamo a programmare in TI BASIC

- Oltre 90 esercizi risolti e proposti.
- Tutte le funzioni del TI BASIC arricchite con molti esempi.
- Grafica e Colore.
- Suono e relative applicazioni.
- Introduzione alle tecniche di programmazione.
- Introduzione ai giochi con il TI 99/4A.
- Oltre ben 10 programmi completi.

Con il patrocinio del Computer Club TI 99

Essemmeci Editore

Mauro Perotti Gabriella Cangemi

Impariamo a programmare in TI BASIC

Con il patrocinio del Computer Club TI 99

Essemmeci Editore

stampa tipo-lito Paglia - Ciampino - tel. 6119890

PREFAZIONE

Lo scopo di questo volume è quello di aiutare l'utente dell'Home Computer TI 99/4A in uno studio più approfondito del TI BASIC e delle sue possibilità.

Al lettore non sono richiesti particolari requisiti ne una conoscenza della programmazione o di particolari nozioni matematiche. Gli argomenti sono trattati in modo volutamente elementare e questo affinché il testo possa essere capito da un pubblico il più vasto possibile.

Il volume consta di tre parti per un totale di 11 capitoli. La prima parte verte sul Basic cosiddetto generale perché si adatta a qualunque macchina programmabile in questo linguaggio; la seconda tratta quelle che sono le speciali subroutines del TI 99/4A e che riguardano essenzialmente la grafica il colore ed il suono. La terza ed ultima parte introduce alle tecniche di programmazione e presenta una serie di applicazioni direttamente eseguibili sul suddetto calcolatore.

Nel tèsto sono compresi un gran numero di esempi ed alla fine di ogni capitolo (tranne l'ultimo) si possono trovare esercizi risolti ed esercizi proposti. Quest'ultimi vanno studiati con cura e quindi eseguiti sulla macchina.

Bisogna inoltre, che dopo i primi capitoli, si provi a stendere qualche programma personale e che sia il frutto della teoria studiata fino a quel punto. È solo in tal modo che si può verificare direttamente la propria preparazione.

La maggior parte dei programmi e problemi presentati non sono particolarmente complessi ma piuttosto specificativi (alcuni programmi sono stati volutamente non ottimizzati affinché potessero risultare i più chiari possibile); è bene quindi che l'utente li approfindisca e cerchi di entrare nella logica con cui sono stati realizzati.

Desideriamo qui ringraziare l'ESSEMMECI ed il COMPUTER CLUB TI 99 ed in particolar modo Marcello Gazzani, Maurizio Manelfi e Gianni Cotugno per l'incoraggiamento che ci hanno dato contribuendo in questo modo alla riuscita di questo volume.

Gli autori

INDICE DEGLI ARGOMENTI

INTRODUZIONE

I.1 **Panoramica sui calcolatori**

Il primo calcolatore elettronico fu realizzato intorno al 1946. Da allora si è aperto un nuovo settore che ha avuto, ed ha tuttora, una enorme e rapidissima evoluzione mai riscontrata in nessun'altra attività della scienza e della tecnica. Per dare un'idea approssimativa di questa evoluzione, soprattutto a coloro che non sono addentro ai problemi di elettronica e d'informatica ad alto livello, ricapitoleremo brevemente qual'era la struttura di un elaboratore agli albori della sua storia, cioè circa trentacinque anni fa, e cercheremo di far comprendere quali sono le tappe significative che hanno permesso un sì rapido sviluppo.

Un calcolatore della prima generazione aveva una memoria costituita da valvole termoioniche (o tubi elettronici). Queste valvole consentivano di realizzare circuiti bistabili che permettevano di memorizzare un'informazione binaria. Questi primi calcolatori, però, avevano un gran numero di inconvenienti: elevati costi di realizzazione, bassa affidabilità dei componenti, notevole ingombro, grande consumo di energia, ecc.

Con la seconda generazione, collocabile verso la fine degli anni 50, si hanno due importanti innovazioni: la nascita dei dipositivi allo stato solido (transistor) per quanto riguarda l'«Hardware» e la nascita dei linguaggi simbolici per ciò che riguarda il «Software». I tubi elettronici furono sostituiti dai transistor; ciò comporta enormi vantaggi sotto tutti i punti di vista: il transistor è nettamente più piccolo, dissipa pochissima energia, consente velocità di calcolo più elevate ed infine è cento volte più affidabile della valvola.

La terza generazione ha come data di riferimento la metà degli anni 60 e presenta innovazioni importanti sia nell'Hardware che nel Software. I circuiti integrati prendono il posto dei transistor, con ulteriori vantaggi sia nel costo che nella velocità. Tra i dispositivi periferici nasce, con l'introduzione dei dischi magnetici, il rivoluzionario concetto dell'accesso diretto. È possibile, quindi, rintracciare in pochi millisecondi un'informazione particolare tra milioni d'informazioni.

La distinzione tra la quarta e la terza generazione non è così evidente come per le precedenti; alcuni la fanno coincidere con la nascita dell'IBM 370, altri con la realizzazione tecnica dei microprocessori. Un microprocessore è un unità centrale (CPU: Central Processor Unit) completa, realizzata su un unico circuto integrato. Il primo processore fu una CPU a 4 bit, e comparve sul mercato nel 1971, rapidamente seguito da microprocessori ad 8 bit. Attualmente esistono anche a 16 bit. Il numero di bit di un microprocessore è un indice della sua potenza di calcolo; a parità di altre funzioni, un numero maggiore di bit è sinonimo di maggiore potenza.

Il TI 99/4A, l'APPLE II e lo SPECTRUM sono esempi di calcolatori della quarta generazione.

I.2 **Classificazione dei linguaggi di programmazione**

Per comunicare con un calcolatore, attraverso la scrittura di un programma, dobbiamo usare un linguaggio ch'esso sia in grado di capire e che noi dobbiamo imparare.

Il «linguaggio macchina», formato da cifre binarie, è quello usato dal calcolatore per la rappresentazione interna dei dati. Se l'uomo dovesse sempre usare questo linguaggio per comunicare con il calcolatore sarebbe un'impresa quasi disperata: un messaggio codificato in binario è lungo, noioso e difficile da scrivere. Esistono invece linguaggi simbolici di programmazione che consentono di scrivere un programma con parole del linguaggio corrente, in genere inglese, e con il sistema di numerazione decimale.

Tra questi l'ASSEMBLER è il linguaggio simbolico, orientato più verso il calcolatore che verso il programmatore, in cui praticamente si ha una corrispondenza uno ad uno tra istruzioni simboliche ed istruzioni in linguaggio macchina. Esso viene usato prevalentemente da personale specializzato addetto alla gestione dei centri di calcolo.

I linguaggi simbolici, detti anche evoluti, sono più orientati verso l'uomo che verso la macchina ed hanno un diverso rapporto tra istruzioni simboliche ed istruzioni macchina: una istruzione simbolica riassume, sinteticamente ed in forma comprensibile all'uomo, diverse istruzioni del linguaggio macchina.

Accenniamo ora ad alcuni linguaggi più comunemente usati.

Il FORTRAN ed il COBOL sono due linguaggi formulati molti anni fa e sono usati rispettivamente per applicazion scientifiche e gestionali. Durante questi anni hanno subito vari perfezionamenti fino a diventare utili e comodi da usare.

Il PASCAL fu sviluppato come strumento per applicare buone tecniche di programmazione.

Il BASIC è un linguaggio di programmazione semplice ma efficiente, adatto ad utenti anche poco esperti. Le istruzioni di un programma BASIC assomigliano molto alle solite operazioni matematiche e a semplici frasi della lingua inglese. È un linguaggio interattivo, cioè consente un colloquio diretto tra utente e calcolatore semplificando molto il lavoro di programmazione. Queste sono tutte caratteristiche che lo rendono particolarmente adatto ai «Personal Computer» che stanno avendo così larga diffusione su tutti i mercati mondiali.

I.3 **Mini e Microcalcolatori**

I minicalcolatori devono il loro nome alle ridotte dimensioni (posson infatti essere costruiti attorno ad una sola scheda elettronica grande come un foglio protocollo) ed al loro costo relativamente contenuto. Essi permettono la programmazione in differenti linguaggi (FORTRAN, PASCAL, BASIC, COBOL,

ecc.) a seconda degli usi a cui sono destinati. Il costo e la flessibilità d'impiego li rendono adatti ad applicazioni che vanno dalla gestione amministrativa di piccole e medie aziende, al controllo dei processi industriali e di macchine utensili ed al calcolo tecnico scientifico.

Con il termine microcalcolatore si intende invece un sistema composto da una CPU (realizzata attraverso un microprocessore), una memoria di programma (ROM), una memoria per l'immagazzinamento dei dati e dei programmi (RAM), circuiti di I/O (Ingresso-uscita) e un generatore di clock.

I vantaggi offerti da microcalcolatori rispetto ai mini sono un prezzo ancora più basso ed un minore ingombro. Di contro, però, il numero di bit per istruzione è ancora oggi minore o uguale a 16 anche se si stanno studiando microprocessori a 32 e perfino a 64 bit.

I.4 **Introduzione al TI 99/4A**

Il microcalcolatore TI 99/4A è un Home Computer (computer da casa) la cui CPU è realizzata da un microprocessore a 16 bit TMS 9900. La memoria RAM è di 16 kbyte (1 kbyte = 1024 byte e 1 byte = 8 bit) ed è espandibile fino a 48 kbyte.

Permette l'utilizzo di periferiche quali i floppy-disk e registratori a cassetta di tipo standard per la memorizzazione permanente di dati e programmi. Esso è direttamente programmabile in TI BASIC che è uno tra i più semplici linguaggi per l'utente. Nel caso in cui l'utente abbia esigenze di programmazione più sofisticate, è previsto l'inserimento di moduli che permettono la programmazione in UCSD PASCAL, in ASSEMBLER ed in EXTENDED BASIC (una forma più potente del suddetto TI BASIC).

CAPITOLO 1

Primi approcci con il TI 99/4A

Questo libro è stato ideato per aiutare coloro che intendono imparare il TI BASIC usando l'Home Computer TI 99/4A sfruttando appieno tutte le sue possibilità. Tuttavia, molte istruzioni che fanno parte di questo linguaggio possono essere usate su qualsiasi computer programmabile in BASIC; il TI BASIC, infatti, è in linea con la convenzione standard americana per il minimo BASIC, la quale stabilisce che tutti i computer programmabili in tale linguaggio devono comunque possedere un comune denominatore per ciò che riguarda le istruzioni fondamentali.

Prima di affrontare lo studio vero e proprio del TI BASIC, è necessario prendere un po' di confidenza con il calcolatore per imparare a comunicare con lui per ciò che riguarda l'essenziale.

Cominciamo quindi, per prima cosa, a familiarizzare con la tastiera.

1.1 **Tastiera**

La tastiera del TI 99/4A è evidentamente simile ad una normale macchina da scrivere di tipo QWERTY. Osservandola meglio, però, ci si accorge che oltre ai normali caratteri ce ne sono altri speciali che per il momento sono a voi poco conosciuti. Alcuni di essi, di cui si vedrà in seguito l'utilizzo, sono detti tasti di funzione.

I tasti numerici e quelli di punteggiatura portano inciso sulla parte frontale un altro simbolo. Quando normalmente viene battuto uno di questi tasti, il carattere che compare sullo schermo del monitor è quello inferiore. Qualora si voglia adoperare il carattere impresso superiormente si deve digitarlo contemporaneamente al tasto SHIFT.

Oltre ai caratteri sinora menzionati ve ne sono altri incisi sul lato di alcuni dei tasti. Per usarli dobbiamo premerli contemporaneamente al tasto funzione FCTN che si trova sulla destra della barra spaziatrice.

Il tasto FCTN, associato ad altri, ha però anche altre funzioni; le riportiamo qui di seguito.

FCTN 1 (DEL) Questi due tasti insieme permettono di cancellare una lettera, numero o altro carattere nella linea che si sta scrivendo. Posizioniamo il cursore sul carattere da cancellare e battiamo contemporaneamente FCTN e 1; oltre alla cancellazione si può notare che si provoca uno slittamento della restante frase verso sinistra, in modo che la scrittura resti compatta.

FCTN 2 (INS) In questo modo si inserisce uno o più caratteri nella linea che si sta scrivendo. Dopo aver battuto questi due tasti contemporaneamente non succede nulla di visibile, ma se si batte una qualsiasi lettera dopo si nota ch'essa appare dove è posizionato il cursore ed il resto della frase slitta tutto verso destra.

FCTN 3 (ERASE) Premendo questi due tasti insieme si provoca la cancellazione di tutta l'ultima linea scritta.

FCTN = (QUIT) Con questo abbinamento si cancella completamente ogni dato o programma o informazione sia dal video che dalla memoria.

FCTN 4 (CLEAR) Premendo questi due tasti si interrompe lo «scrolling» del «listato» (il significato di questi due termini è riportato sul glossario) di un programma. Quando si premono, però, durante l'esecuzione di un programma, permettono d'interromperlo ed il calcolatore, fermandosi, indica all'utente la linea successiva a quella appena eseguita con un messaggio audiovisivo del tipo: BREAKPOINT AT XX (dove XX indica il numero di linea in questione).

FCTN S Questa combinazione permette di spostare il cursore di una posizione verso sinistra. Quando il cursore, spostandosi, passa su caratteri già battuti non li cancella né li modifica.

FCTN D Ha la stessa funzione del FCTN S tranne che per la direzione che in questo caso è verso destra.

Il tasto forse più importante, e comunque quello più usato, è il tasto ENTER. Esso, in un certo senso, si può paragonare al tasto di ritorno di una normale macchina da scrivere; battendo ENTER, infatti, il cursore si sposta inferiormente di una riga e si pone al margine sinistro. Ma la sua funzione più importante è quella di far accettare al computer la riga di programma che si è appena battuta, cioè di far «entrare» un informazione in memoria o di far eseguire un comando diretto.

Vogliamo, ora, sottolineare che tutti i tasti del TI 99/4A sono ripetitivi, infatti, supponendo di voler stampare lo stesso carattere più volte consecutivamente, è sufficiente tener premuto per più di un secondo il tasto ad esso corrispondente.

Esaminiamo infine il tasto ALPHA LOCK e la barra spaziatrice. Il primo serve per abilitare la tastiera a scrivere in maiuscolo le lettere dell'alfabeto; la seconda è analoga a quella di una normale macchina da scrivere ma se sposta il cursore su caratteri preesistenti, essi vengono cancellati.

1.2 **Aritmetica del calcolatore**

COSTANTI NUMERICHE

In BASIC, come in matematica, una costante numerica (numero) può es-

sere intera o decimale. Essa si scrive utilizzando direttamente i numeri sulla tastiera con l'accortezza di sostituire la virgola decimale con il punto. Quando però si lavora con numeri molto grandi oppure molto piccoli, si deve adottare la cosiddetta notazione scientifica. Essa consiste nel rappresentare il numero come prodotto di una base (mantissa) per un'opportuna potenza di dieci.

Esempi di numeri scritti in notazione scientifica sono:

32640 =	$3,264 \times 10^{4}$	in BASIC si scrive	3.264E4
-0,81 =	$-8,1 \times 10^{1}$	»	-8.1E-1
	$2,512 \times 10^{-15}$	»	2.512E-15
	$-7,31 \times 10^{18}$	»	-7.31E18

COSTANTI ALFANUMERICHE

Quando la stampa di una frase ricorre più di una volta in un programma (per la definizione di programma vedi glossario) è comodo considerare quella frase come una costante e rappresentarla interamente con un opportuno simbolo che può essere richiamato quando occorre. Questo tipo di costante viene chiamata alfanumerica o a stringa. Nell'esempio che segue mostriamo come si può assegnare ad un opportuno simbolo una costante alfanumerica:

```
10 A$; "BUONGIORNO!"
20 PRINT A$



100 PRINT A$
```

La parola chiave PRINT sarà meglio illustrarla in seguito, ma per la comprensione di questo esempio è sufficiente sapere ch'essa fa stampare su monitor il valore della costante o della variabile, o la frase da cui è seguito.

Ogni volta che con l'istruzione PRINT richiamiamo il simbolo A$ sul video comparirà la frase BUONGIORNO!

A$ comunque non è una scelta casuale! Infatti il simbolo con cui identifichiamo una costante alfanumerica deve sempre avere come primo carattere una lettera e come ultimo il segno $.

È importante notare che in questa assegnazione la costante alfanumerica deve essere sempre racchiusa tra virgolette (vedi linea 10).

In sostanza una stringa o costante alfanumerica è una sequenza di caratteri (lettere, numeri, caratteri speciali) fra virgolette. Anche gli spazi bianchi, in una costante alfanumerica, non vengono ignorati, ma sono considerati facenti parte della stringa stessa. Ad esempio la stringa "CAPO STAZIONE" è diversa dalla stringa "CAPOSTAZIONE".

VARIABILI

Come le costanti così anche le variabili possono essere di due tipi: numeriche o alfanumeriche. Ambedue i tipi devono essere rappresentate da una sequenza di caratteri (fino ad un massimo di 15) di cui il primo deve necessariamente essere una lettera oppure un particolare simbolo speciale (/,-,],[,@,)
Le variabili alfanumeriche a differenza di quelle numeriche hanno sempre come ultimo carattere $.

Esempi di variabili numeriche sono:

A CX /XC BA513C PIPPO F1 C4 -ZE3 F14CX

Quelle che seguono sono invece variabili alfanumeriche:

A$ ABCD$ -CASA$ C134S$ PIPPO$ /MG$ HGY76/-$

OPERATORI ARITMETICI

Vengono definiti in questo modo alcuni simboli speciali che individuano operazioni aritmetiche:

per l'addizione	+
per la sottrazione	-
per la moltiplicazione	*
per la divisione	/
per l'elevaz. a potenza	∧

Gli operatori collegano numeri e formule costituendo così le espressioni aritmetiche.

Per esempio vediamo alcune formule e relativa traduzione in BASIC:

$(a+b-c)+d$	(A+B-C)+D
$\frac{2x-3y}{u+v}(a+b)^2$	((2*X-3*Y)/(U+V))*(A+B)∧2
$3,14\ r^2$	3.14*R∧2

GERARCHIA DELLE OPERAZIONI

Quando più operatori compaiono nella stessa formula, possono sorgere dei dubbi sulsignificato delle operazioni. Per esempio, la formula 4*A-3*B corrisponde al termine algebrico (4a)-(3b) oppure a 4(a-3b)? Analogamente, X/Y*Z corrisponde ad x/(yz) oppure ad (x/y)z? È facile superare questo dubbio quando si conosce l'ordine con cui vengono eseguite le operazioni nell'ambito di una data espressione algebrica.

La gerarchia delle operazioni in BASIC è la stessa usata in matematica ed è la seguente:

1. Elevazione a potenza. Tutte le operazioni di questo tipo vanno eseguite per prime.

2. Moltiplicazione e divisione. Esse hanno la stessa priorità e vengono eseguite dopo l'elevazione a potenza.

3. Addizione e sottrazione. Si eseguono per ultime.

In un assegnato gruppo gerarchico le operazioni vengono eseguite da sinistra verso destra; ciò significa che tra due operazioni di medesima priorità viene eseguita per prima quella che sta a sinistra. Ad esempio la formula A/B*C è la equivalente in BASIC dell'espressione (a/b)c. Analogamente B∧2-4*A*C rappresenta la formula b^2-(4ac).

PARENTESI IN BASIC

In una formula a volte è necessario alterare la normale gerarchia delle operazioni e questo lo si ottiene con l'uso delle parentesi. Quando si debbono risolvere espressioni racchiuse in più d'una coppia di parentesi, del tipo ad esempio (...(..(...).).), saranno eseguite per prime le operazioni racchiuse nella coppia più interna, seguite da quelle contenute nelle parentesi successive e così via. In una data coppia di parentesi, comunque, vale sempre la normale gerarchia. Bisogna ricordarsi di usare sempre *coppie* di parentesi: un errore piuttosto comune è quello di aprire più parentesi di quelle che si chiudono o viceversa. Questo tipo di errore è anche favorito dal fatto che mentre in matematica esistono diversi tipi di parentesi, in BASIC si usa solo quella tonda. Vediamo ora alcuni esempi.

Supponiamo di volter tradurre in BASIC la seguente espressione matematica:

$$\{[2(a+b)^2+(3c)^2]^m\}^{(n+1)}$$

essa corrisponde a:

((2*(A+B)∧2+(3*C)∧2)∧M)∧(N+1)

L'espressione

a/b + c/d

è tradotta in BASIC in

A/B + C/D oppure (A/B) + (C/D)

Osserviamo che le due espressioni sono entrambe corrette; naturalmente la seconda espressione è ridondante per quanto riguarda le parentesi, quelle in più vanno, se è possibile, eliminate per questioni di spazio in memoria e tempi di esecuzione; ma in caso di dubbio è sempre meglio usarle.

N.B. Se la base di una potenza è negativa, l'esponente può essere solo un numero intero quando l'operazione è eseguita da programma. Questa restrizione è giustificata se si conosce la procedura con cui la macchina compie l'elevazione a potenza. Se l'esponente è una grandezza intera, il calcolatore ottiene il risultato moltiplicando la base per se stessa tante volte quanto è l'esponente.
Se l'esponente è decimale, la procedura invece è la seguente: la macchina calcola il logaritmo della base, moltiplica per l'esponente ed infine calcola l'antilogaritmo del risultato così ottenuto. Ora, poiché il logaritmo di un numero negativo non è definito, se la base è negativa il calcolatore dà errore. Ciò non accade se l'operazione viene eseguita con istruzioni dirette poiché il calcolatore segue un'altra procedura.

ESERCIZI RISOLTI (CAP. 1)

1) Abbiamo visto che sia la barra spaziatrice che l'abbinamento dei tasti FCTN e D provoca lo spostamento del cursore verso destra. Ma allora qual'è la differenza?

Soluzione: La differenza è che mentre FCTN e D muove il cursore verso destra lasciando inalterati i caratteri sopra cui passa, la barra spaziatrice, invece, li cancella dal video.

2) Quando digitiamo contemporaneamente FCTN e= (QUIT) che cosa accade?

Soluzione: Il quadro viene riportato allo stato iniziale (quello, cioé, che il calcolatore mostra quando viene acceso). Contemporaneamente si ha l'azzeramento di tutte le memorie e quindi, qualunque informazione contenuta in memoria, viene irrimediabilmente cancellata.

3) Risolvere le seguenti questioni concernenti l'aritmetica del calcolatore:

a) esprimere le seguenti grandezze numeriche in forma TI BASIC: 9210; -128; $201{,}374 \times 10^5$; 0-0,00000952; 51×10^{-19};

b) identificare eventuali errori nelle seguenti grandezze numeriche scritte in forma TI BASIC:

59,32   31.12E-2   -+511;

c) tra le seguenti variabili numeriche ve ne sono alcune non corrette. Identificarle motivando il perché:

J9   B$ 1A7   A29

d) nello spirito dell'esercizio precedente, verificare le seguenti variabili a stringa (o alfanumeriche).

C$ A   Q$1   7A$

e) tradurre in forma TI BASIC le seguenti espressioni algebriche:

$x^2 + y^2 - z^2$; $4y-1$; $(5z + x)\ y$; $(i + j)^2$

Soluzione:

a) 9210 oppure 9.21E3; -128 oppure -1.28E2; 201.374E5; -952E-8; 51 E-19 oppure 5.1E-18

b) 59,32 è una scrittura scorretta in quanto la virgola decimale non è ammessa; +-511 è anch'essa una scrittura scorretta in quanto il doppio segno non è ammesso.

c) B$ non è corretta perché l'ultimo carattere di una variabile numerica non può essere un $;
1A7 non è corretta perché il primo carattere di una qualunque variabile non può essere un numero;

d) A non è corretta perché l'ultimo carattere di una variabile alfanumerica deve essere necessariamente un $;
Q$1 non è corretta per il motivo precedente ed in più perché il carattere $ non può comparire all'interno di una variabile; 7A$ non è corretta perché il primo carattere di una qualunque variabile non può essere numerico;

e) X∧2+Y∧2+Z∧2   4*Y-1   (5*Z+X)/Y   (I+J)∧2

4) Verificare se le seguenti traduzioni sono corrette ed in caso contrario specificarne il motivo:

$2(x+y)$	2*(X+Y)
$3x+y$	3*(X+Y)
$(i+j-l)^m$	I+J-L∧M

Soluzione: la prima è corretta. La seconda è scorretta perché tale traduzione corrisponde al termine algebrico $3(x+y)$. La traduzione corretta è invece 3*X+Y. Anche la terza è scorretta: mancano infatti le parentesi. La traduzione corretta è (I+J-L)∧M.

5) Identificare eventuali errori nelle espressioni TI BASIC che seguono:

a) X*(Y+Z*(3*X-2)∧2

b) 3X-2*Y+5*(X-Z)∧2

c) (X+Y)∧2-(X-Y)∧2

Soluzione:
a) scorretta: il numero delle parentesi aperte non è uguale al numero delle parentesi chiuse (o come si suol dire le parentesi non son bilanciate);

b) scorretta: nel primo termine manca un qualche operatore tra la variabile X e la costante numerica 3;

c) corretta.

ESERCIZI PROPOSTI (CAP. 1)

1) Illustrare dettagliatamente la differenza tra un numero scritto in notazione scientifica e lo stesso numero scritto in forma decimale o intera.

2) Qual'è la gerarchia naturale tra le operazioni in TI BASIC?

b) All'interno della stessa parentesi, viene effettuata prima l'operazione di moltiplicazione o prima quella di divisione?

c) Quando si vuole elevare ad una certa potenza la somma di due o più variabili, è necessario racchiuderle tra parentesi?

3) Può una grandezza negativa essere elevata ad un esponente non intero? Se la risposta è no, spiegare come si può evitare l'ostacolo.

4) Tradurre in forma TI BASIC le seguenti espressioni algebriche:

$2x + y - z^2$ $\quad (m+n)^{\frac{I-1}{2}}$ $\quad \frac{a}{b} + \frac{c}{d} - 1$

5) Identificare fra le seguenti variabili TI BASIC quali sono numeriche, quali a stringa e quali sono errate

A$ A1 C$T PIPPO1

CAPITOLO 2

Comandi diretti

Per comando diretto o immediato s'intende un'istruzione che, una volta digitata, viene immediatamente eseguita dal calcolatore dopo aver battuto il tasto ENTER.

Per esempio, proviamo a scrivere:

```
PRINT "IL MIO NOME È MARIO ROSSI"
```

e quindi battiamo ENTER

Che cosa accade? Sul video viene stampata immediatamente la frase tra virgolette.

Un comando si dice indiretto, invece, quando è inserito in un programma. Vediamo, ad esempio, come il PRINT (che come abbiamo già accennato vuol dire stampa) può essere usato anche da programma e cioè come comando indiretto:

```
10 PRINT "IL MIO NOME È MARIO ROSSI"
20 END
```

Questo, anche se estremamente elementare, è un esempio di programma. Quindi, per programma si intende una sequenza ordinata d'istruzioni contraddistinte da un numero compreso tra 1 e 32767, detto numero di linea, che serve ad indicare al calcolatore l'ordine con cui vanno eseguite le istruzioni.

Per mandare in esecuzione un programma, o per farlo girare, dobbiamo digitare la parola RUN e battere ENTER.

Vorremmo ribadire, ancora una volta, che alcuni comandi possono essere usati come istruzioni per un programma, ed alcune istruzioni possono essere usate anche come comandi immediati: uno fra questi è appunto lo 'statement' PRINT.

Passiamo ora ad illustrare i principali *comandi diretti (o immediati).*

2.1 NEW

Questo comando serve per cancellare tutto ciò che è contenuto in memoria. È buona norma usare questo comando prima di cominciare a scrivere un nuovo programma.
ATTENZIONE! Si deve stare in guardia a non usare il NEW fuori luogo: si potrebbe, infatti, cancellare involontariamente dalla memoria il programma o una parte di esso faticosamente battuta e dover ricominciare tutto da capo!!

Per ciò che riguarda il TI 99/4A, osserviamo che la esecuzione di questo comando provoca la cancellazione di tutti i caratteri presenti sul video e la comparsa del messaggio «TI BASIC READY». A questo punto il calcolatore è pronto per accettare un nuovo programma.

2.2 RUN

Questo comando serve per mandare in esecuzione un programma che si trova in memoria. Se scriviamo solo RUN, il calcolatore esegue il programma partendo dalla linea con il numero più piccolo. Se si vuole che il calcolatore inizi ad eseguire il programma da una certa linea in poi, bisogna scrivere RUN n dove n indica la linea in questione.

Se proviamo, però, a dare il comando RUN quando in memoria non c'è nulla, compare il messaggio: CAN'T DO THAT, che ci avverte che in memoria non vi sono programmi da eseguire.

Ad onor di completezza vogliamo informarvi che in alcune macchine della quarta generazione è possibile usare il comando RUN anche come istruzione (ciò è possibile, per esempio, nell'EXTENDED BASIC). Non ci occupiamo però dei suoi effetti in quanto non è competenza di questo volume.

2.3 LIST

Serve per 'listare' sul video il programma che è in memoria. Può essere usato da solo o insieme al segno - (meno) ed a un numero di linea.

Usato da solo, produce la lista di tutto il programma con eventuale 'scrolling' se le linee di programma sono tante da non poter essere visualizzate tutte insieme sul video LIST -n serve per listare tutte le linee con numeri minori e uguali a quello specificato da n; usando LIST n- vengono listate tutte le linee con numero maggiore e uguale a n.

Possiamo anche scrivere LIST n-m; in questo caso vedremo scorrere sul video le linee di programma comprese tra n ed m. Digitando invece la frase

LIST n, vedremo comparire sul video la linea di programma numero n.

Vediamo ora qualche esempio che racchiude questi tre importanti comandi. Scriviamo:

```
100 A=5
110 B=-3
120 C=A+B
130 PRINT C
140 STOP
150 END
```

Senza preoccuparci molto di quello che abbiamo fatto, battiamo RUN e poi ENTER. Osserviamo che sul video compare il numero 2: abbiamo cioè mandato in esecuzione il programma appena battuto. Ora scriviamo LIST e battiamo il tasto ENTER. Vedremo comparire sul video, una ad una, tutte le istruzioni che abbiamo precedentemente scritto. È importante ora che l'utente si eserciti ad applicare tutto ciò che è stato detto in questo paragrafo, usando il programma appena visto o (se è in grado di farlo) inventandone altri.

2.4 **BYE**

Questo comando avverte il calcolatore che abbiamo finito di lavorare con lui e siamo in procinto di spegnere la macchina.

Quando al calcolatore viene dato il BYE, esso cancella tutti i programmi eventualmente residenti in memoria centrale. È buona abitudine, anche se non è obbligatorio, dare il BYE prime di spegnere l'interruttore generale del calcolatore.

2.5 **NUMBER (abbreviato NUM)**

Abbiamo detto che, quando si vuole scrivere un programma, si deve numerare ogni istruzione in modo che il calcolatore sappia qual'è il giusto ordine d'esecuzione.

Eseguire questa operazione per ogni linea è certamente una cosa noiosa. Perciò è stata prevista (non però su tutti i microcalcolatori) l'esistenza di un sottoprogramma che provvede automaticamente a questa numerazione successiva: esso viene richiamato dalla parola chiave NUMBER.

Nel caso del TI 99/4A, questo comando genera automaticamente una numerazione che parte da 100 ed incrementa il numero di linea di 10. È possibile, comunque, specificare sia la linea iniziale che l'incremento facendo seguire NUMBER da due numeri interspaziati dalla virgola che indicano rispettivamente: il primo, il numero che deve identificare la prima linea; il secondo l'incremento. Ad esempio se scriviamo: NUMBER 10,5 il calcolatore numererà la linea iniziale con 10, la seconda con 15, la terza con 20 e così via.

È altresì possibile specificare solo la linea iniziale e in questo caso è sottointeso che l'incremento è 10. Si può anche specificare solo l'incremento sottointendendo che la linea iniziale è 100.

Per esempio:

NUMBER 10	(parte da 10 e va di 10 in 10)
NUMBER 5	(parte da 100 e va di 5 in 5)

Per gli utenti del TI 99/4A è utile sapere che per annullare la numerazione automatica (se per esempio si vuole continuare usando la numerazione manuale riga per riga) è necessario battere due volte il tasto ENTER alla fine dell'ultima istruzione digitata.

Da notare che, sempre per la suddetta macchina, è prevista l'abbreviazione di questo comando in NUM.

2.6 **RESEQUENCE (abbreviato RES)**

È buona abitudine di un programmatore, quando scrive un programma, lasciare tra un numero di linea ed il successivo un certo intervallo disponibile (di norma 10 numeri). Questo perché se, in seguito, è necessario inserire tra due linee di programma una terza, si hanno dei numeri di linea liberi da poter utilizzare senza dover riscrivere tutto il programma da capo. A volte, dopo aver aggiunto molte linee ad un programma, ci si trova a non aver più spazio per un ulteriore inserimento; per questo e per altri motivi esiste in TI BASIC un comando che fa eseguire alla macchina una rinumerazione automatica a partire da una data linea prefissata e con un certo incremento. Il comando è: RESEQUENCE n,m oppure RES n,m dove n rappresenta la linea iniziale e m l'incremento. Nel caso in cui RES venga dato senza argomenti n ed m il calcolatore assegna ad n il valore 100 e ad m 10 automaticamente.

2.7 **CONTINUE (abbreviato CON)**

Capita a volte di dover interrompere l'elaborazione di un programma per diversi motivi (spesso di controllo). Per riprendere l'elaborazione dall stesso punto in cui era stata arrestata, bisogna usare il comando CON (se utilizzassimo invece RUN, il calcolatore riprenderebbe l'esecuzione dalla prima linea di programma e questo non è sempre necessario).

2.8 **EDIT**

Questo comando serve per correggere uno o più errori in una linea di programma già battuta ed accettata dal calcolatore ed è sempre seguito dal numero della linea che si vuole correggere. Battuto quindi EDIT, il numero di linea ed ENTER, appare sullo schermo la linea da modificare.

A questo punto, è necessario parlare di due tasti speciali del TI 99/4A che sono stati volutamente ignorati fino a questo punto.

Premendo contemporaneamente FCTN ed S oppure FCTN e D, siamo in grado di spostare il cursore di una posizione rispettivamente verso sinistra o verso destra. Il cursore, spostandosi, non altera alcun carattere; in tal modo è possibile operare la correzione come spiegato nel paragrafo «tastiera». Una volta entrati in modo «EDIT» è possibile effettuare correzioni su più linee usando i tasti FCTN ed E e FCTN ed X. In tal modo, rispettivamente, vengono visualizzate le linee immediatamente inferiori e superiori a quella presente sul video permettendo così la correzione.

Spesso, quando si passa molto tempo davanti al calcolatore, è possibile trovare delle particolarità intrinseche della macchina che neppure il manuale stesso nomina.

Per esempio, per correggere le linee errate di un listato, c'è un modo più rapido che evita di dover scrivere ogni volta il comando EDIT. Per entrare automaticamente in modo «EDIT» dobbiamo solo scrivere il numero della linea interessata e quindi battere contemporaneamente i tasti FCTN e X oppure FCTN e E (consigliamo di tener premuto il tasto FCTN e poi premere il tasto X od E fino a che la linea non sia comparsa sul video).

Esiste inoltre un altro modo che permette di «editare»: usare il comando «NUMBER». Richiamando NUM ed il numero di linea, è possibile correggere l'istruzione che viene visualizzata e tutte le linee immediatamente successive (che vengono a loro volta visualizzate premendo ancora ENTER). Non si può, però, chiedere alla macchina di visualizzare (con il tasto FCTN ed X) la linea immediatamente superiore a quella attualmente sul monitor (ossia quella con numero di linea più basso) come era possibile fare con l'EDIT e questo costituisce senz'altro un piccolo svantaggio.

Di solito si usa il comando NUM quando oltre che a correggere una o più linee di un programma, se ne vogliono aggiungere delle altre; in questo caso è molto più comodo del comando EDIT.

2.9 **SAVE**

Questo comando serve per trasferire un programma che si trova in memoria su cassetta o su floppy-disk.

Nel caso di memorizzazione su cassetta il comando è: SAVE CS1 o SAVE CS2, a seconda che si voglia utilizzare rispettivamente il registratore 1 o il 2 (infatti, come ben sapere, con il TI 99/4A è possibile lavorare con due registratori contemporaneamente).

Se vogliamo invece memorizzare un programma su disco, il comando da scrivere è: SAVE DSK1.nome del programma per l'unità a disco 1 e SAVE DISK2.nome del programma per l'unità a disco 2.

N.B.: E importante non omettere mai il punto prima di scrivere il nome del programma.

2.10 **OLD**

Serve per caricare in memoria un programma già inciso su nastro o su disco. Il comando completo che dobbiamo dare è: OLD CS1 oppure OLD DSK1.nome programma o anche OLD DSK2.nome programma (la cassetta CS2 non è stata menzionata poiché il secondo registratore serve solo per registrare).

A questo punto il sistema visualizza sullo schermo le istruzioni che dobbiamo compiere con il registratore e quindi carica in memoria il primo programma che si trova sul nastro o quello richiamato da disco.

Può accadere che il calcolatore non abbia caricato correttamente il programma in memoria ed allora provvede ad avvertire del fatto l'utente con un messaggio audiovisivo. In questo caso di deve controllare che tutti i collegamenti tra registratore e computer siano a posto e che il volume ed i toni si trovino circa a metà della loro massima escursione. Verificato ciò si può procedere a ripetere l'operazione.

2.11 **Identificazione e correzione degli errori**

Uno degli argomenti più ardui da trattare è quello degli errori. Innanzitut-

to è difficile scrivere un programma alquanto complicato senza incorrere in errori, e questa è una cosa ben chiara per tutti quelli che s'intendono di programmazione.

Gli errori che si possono commettere sono essenzialmente di due tipi: errori di grammatica o di sintassi ed errori di logica.

Gli errori di sintassi sono quelli che impediscono alla macchina di interpretare il programma (esempi di tali errori sono: parentesi non bilanciate, una frase scritta dopo il comando INPUT non racchiusa tra virgolette, assenza di un numero di linea, ecc.). Errori di questo tipo sono molto comuni e son dovuti, nella maggior parte dei casi, a distrazione da parte dell'utente nel battere il programma. Tali errori, comunque, sono quelli che s'individuano più facilmente poiché è il calcolatore stesso che li segnala emettendo un diagnostico (ossia un messaggio di errore). Tale diagnostico individua sia il tipo di errore che il numero di linea in cui esso si è verificato dando, quindi, un rilevante aiuto al programmatore. Consigliamo, perciò, che l'utente si abitui sin dall'inizio a consultare il paragrafo «Messaggi d'errore» all'interno del manuale del TI 99/4A.

Mentre il calcolatore si accorge degli errori sintattici non si avvede, invece, di quelli logici; pertanto quest'ultimi sono più difficili da identificare e più insidiosi.

Se facciamo girare un programma non affetto da errori sintattici ma con errori di logica, la macchina esegue correttamente tutte le istruzioni ma i risultati che si ottengono non sono quelli voluti.

Per illustrare meglio questo ultimo delicato concetto facciamo un esempio. Supponiamo di voler calcolare la seguente formula introducendo da tastiera i valori per x:

$$y = \frac{2x+3}{x^2} + \frac{3}{4}\left(\frac{2x+3}{x^2}\right)^{2}-3$$

data la particolarità della formula si può pensare di usare una nuova variabile (per semplificare il programma e ridurre i tempi d'esecuzione):

$$u = \frac{2x+3}{x^2}$$

nella quale variabile y sarà espressa nel modo seguente:

$$y = u + \frac{3}{4} u^2 - 3$$

Vediamo ora il programma:

```
100 INPUT "DAMMI IL VALORE DI X"; X
110 U=2*X+3/X*X
120 Y=U*(3/4*U)^2-3
130 PRINT "Y=";Y
140 END
```

Come si può facilmente constatare il programma così battuto non genera diagnostici da parte della macchina una volta dato il comando RUN, e quindi stampa il valore di Y.

A questo punto, se il programmatore non stesse in guardia contro gli errori logici, potrebbe erroneamente supporre la validità del precedente risultato; ma eseguendo semplici calcoli si accorge, invece, che la soluzione così ottenuta per Y non è corretta. Se infatti si risponde all'INPUT con X=2 il calcolatore stampe per Y il valore 31.5625 che chiaramente non coincide con il valore ottenuto manualmente 1.046875. Questa evidente discordanza tra i valori così ottenuti per Y mostra, inequivocabilmente, la presenza di un errore logico (anzi ce n'è più d'uno).

Riprendiamo le linee 110 e 120. Si può notare, riguardando meglio la formula, che nella prima mancano delle parentesi necessarie, e nella seconda ce ne sono due di «troppo» che trasformano il significato dell'espressione stessa. Vediamo, invece, come devono scriversi le suddette linee per essere corrette dal punto di vista logico:

```
110 U=(2*X+3)/(X*X)
120 Y=U+3/4*U^2-3
```

Per stabilire se un programma è affetto da errori logici non ci sono regole fisse da seguire; possiamo qui dare soltanto degli utili consigli. Il resto è affidato all'esperienza ed all'acume del programmatore.

Il primo passo da seguire è quello d'introdurre nel programma dati che producono risultati che si possono controllare (ossia risultati che conosciamo già in partenza). In questo modo, però, anche ottenendo dei risultati uguali a quelli già conosciuti, non è possibile essere assolutamente sicuri che il prgramma sia corretto: alcuni errori, infatti, provocano risultati sbagliati solo in certe circostanza (cioè per determinati valori dei dati in ingresso).

I programmi, quindi, vanno provati più volte e con diversi valori. Quando però i suddetti sono particolarmente complessi, è necessario verificarli con me-

todi diversi come ad esempio la tecnica di «randomizzazione» che usa gruppi di campioni scelti a caso.

Quando si è accertata l'esistenza di un errore logico, è necessario che il programmatore lo trovi per poterlo correggere («debugging logico»: ricerca degli errori logici). Ciò non è sempre facile (anzi non lo è quasi mai!); diamo quindi alcune indicazioni per fare ciò.

Innanzitutto il programmatore deve focalizzare l'attenzione sugli enunciati logici e riguardarli con cura. Se non si riesce in tal modo, può essere utile che l'utente faccia eseguire dalla macchina delle stampe intermedie: normalmente è proprio così che si riescono ad individuare gli errori logici più insidiosi e più «ostinati».

Un'ultima cosa, ma non per questo meno importante, raccomandiamo all'utente: spesso è utile lasciare da parte il programma per un po': è molto frequente, infatti, che l'eccessiva attenzione non permette di vedere al primo tentativo un errore anche ovvio.

A questo punto esaminiamo quali sono i comandi che il TI BASIC mette a disposizione dell'utente per la ricerca degli errori di logica e come si utilizzano.

2.12 **BREAK**

Con il comando BREAK si inseriscono nel programma dei punti di arresto che possono essere utilizzati per trovare gli errori prima menzionati.

Digitando BREAK N (dove N è un numero di linea esistente nel programma) si avverte il calcolatore che dovrà interrompere l'esecuzione non appena avrà terminato di eseguire la linea immediatamente precedente ad N. L'utente si accorgerà di ciò quando il calcolatore avrà stampato sul monitor il messaggio:

BREAKPOINT AT N

È anche possibile interrompere l'esecuzione del programma in più punti diversi con un solo comando BREAK. Per far ciò e sufficiente far seguire il comando dalla lista di numeri di linea voluti. Ad esempio: BREAK 100,125,450; in tal caso il calcolatore interromperà l'esecuzione del programma alla linea immediatamente inferiore alle suddette linee (per riprendere l'esecuzione del programma a partire dalla linea in cui il calcolatore si è interrotto, è necessario digitare il comando CON).

Il comando BREAK può essere usato anche come istruzione da programma. In questo caso possiamo usarlo in due modi diversi:

1). 100 BREAK 200,310,330
2) 120 BREAK

Nel primo caso la macchina fermerà l'esecuzione del programma prima di ogni linea dichiarata nella lista; nel secondo caso fermerà l'esecuzione prima della linea 120 ossia nel punto in cui è situata l'istruzione BREAK.

2.13 **UNBREAK**

Il comando UNBREAK serve per annullare gli effetti prodotti dal comando BREAK (una volta, ad esempio, chè abbiamo terminato il «debugging logico»). Le regole sintattiche a cui deve obbedire sono le stesse del comando BREAK.

È importante far notare che l'effetto di un comando BREAK assegnato come istruzione in un programma e senza lista linee, è possibile annullarlo soltanto togliendo la linea stessa. Ad esempio se inseriamo in un programma l'istruzione:

```
100 BREAK
```

essa avrà effetto ogni volta che daremo il comando RUN e per annullarla dovremo cancellarla dal programma.

Nel caso invece in cui si ha un istruzione del tipo:

```
100 BREAK 200,300,400
```

il comando UNBREAK, dato come comando diretto (cioè senza numero di linea), avrà effetto solo temporaneamente. Ossia se digitiamo il comando nel modo:

```
UNBREAK 300
```

e poi diamo il comando RUN, il calcolatore osserverà i punti di arresto soltanto alle linee 200 e 400. Quando però digitiamo il comando RUN successivamente, il calcolatore tornerà ad osservare nuovamente tutti e tre i punti di arresto precedenti.

2.14 **TRACE**

Vi sono altri due comandi in TI BASIC che aiutano nella ricerca degli errori di logica. Il primo è il comando TRACE. Esso ci consente di vedere l'ordine con cui il calcolatore esegue le istruzioni di un programma. Infatti, dato questo comando in modo diretto, man mano che il calcolatore esegue il programma, appare sul monitor (racchiuso tra parentesi triangolari) il numero della linea attualmente in esecuzione. Vediamo ora un esempio:

```
100 A=12
110 B=45
120 C=A/B
130 PRINT C
```

diamo ora il comando TRACE e facciamo girare il programma: sul monitor apparirà la seguente linea di stampa:

```
<100><110><120><130> 3.75
```

Ogni volta che digiteremo il comando RUN, il calcolatore visualizzerà sempre la stessa linea di stampa: siamo infatti entrati in «modo TRACE».

È possibile inserire il comando TRACE come istruzione nel programma; essa, indipendentemente dal punto in cui è stata posta, agirà come comando immediato.

2.15 **UNTRACE**

Questo comando annulla gli effetti del comando TRACE e si usa sia come comando diretto che come istruzione.

Una cosa molte interessante è l'effetto che queste due istruzioni (TRACE e UNTRACE) hanno sull'esecuzione di un programma, se vengono inserite opportunamente all'interno del medesimo. Se infatti si vuole visualizzare solo un gruppo particolare di istruzioni (all'interno di un programma ben più grande) non è possibile farlo dando il solo comando TRACE poiché, come abbiamo già avuto modo di dire, esso mostra sul video tutte le istruzioni dall'inizio alla fine del programma. Ciò, invece, si può ottenere con il suddetto

abbinamento.

Un esempio chiarirà meglio di qualsiasi frase ciò che bisogna fare. Supponiamo di voler visualizzare soltanto l'esecuzione delle linee che vanno da 300 a 500 e lasciare eseguire normalmente il resto del programma:

```
100 UNTRACE



290 TRACE
300 .........

 ............
500 .........

510 UNTRACE
```

Una simile possibilità è molto utile nei programmi di grafica dove una visualizzazione completa di tutte le linee in esecuzione (ottenuta con il solo comando TRACE), altererebbe la grafica presente sul video.

ESERCIZI RISOLTI (CAP. 2)

1) È possibile, in TI BASIC, usare il comando RUN come istruzione?

Soluzione: No! Ciò è possibile in EXTENDED BASIC.

2) Qual'è la differenza tra questi due comandi diretti
LIST-X LIST X-
dove X è il numero di linea di una linea presente nel programma?

Soluzione: La differenza è che nel primo caso vengono visualizzate tutte le linee il cui numero è minore od uguale a X; mentre nel secondo caso tutte le linee il cui numero è maggiore o uguale a X.

3) Qual'è la differenza tra il comando BYE ed il comando NEW?

Soluzione: Il comando BYE avverte il calcolatore che abbiamo terminato di lavorare con lui e riporta sul video il quadro iniziale: mentre il comando NEW cancella tutto ciò che ha in memoria la macchina fermo restando il modo TI BASIC.

4) Spiegare la differenza tra errori di sintassi ed errori di logica e fare alcuni esempi di tali errori.

Soluzione: I primi sono errori che il programmatore, di norma, compie per distrazione. Tali errori provocano un arresto del programma e la stampa sul video di un diagnostico che aiuta l'utente ad identificare l'errore.

Gli errori di logica sono errori che la macchina non riconosce. Essi, infatti, sono da ricercarsi nella stesura intrinseca dell'algoritmo. Vediamo ora due esempi.

Un errore di sintassi può essere, ad esempio, un bilanciamento non corretto delle parentesi in una formula TI BASIC; un errore di logica può essere, invece, una traduzione non fedele di una formula algebrica. Ad esempio b^2-4ac è tradotta in B∧2-4*A*C. Se però viene tradotta in (B-4*A*C)∧2 viene commesso un errore di logica (in quanto sintatticamente è corretta e la macchina non ferma il programma).

5) I comandi TRACE e BREAK sono comandi implementati nel TI BASIC a quale scopo?

Soluzione: Sono comandi che aiutano nella ricerca degli errori logici (debugging).

ESERCIZI PROPOSTI (CAP.2)

1) Quando viene interrotta l'esecuzione di un programma (con FCTN 4 ad esempio) è possibile far riprendere l'esecuzione dal punto in cui era stata interrotta? Se si in che modo?

2) È possibile caricare in memoria più programmi distinti? (intendiamo con tastiera e non con il registratore)
b) Qual'è l'accortezza che si deve usare per far ciò?
c) Quando si vuole far girare uno di questi, come si deve operare?

3) È possibile dare il comando RUN o LIST quando in macchina non è presente alcun programma?
b) Se lo si fa cosa accade?

4) Spiegare in dettaglio il funzionamento dei comandi NUM e RES e dire se hanno delle particolarità in comune.

5) Qual'è il comando che dobbiamo dare per «salvare» un programma su cassetta? E per salvarlo su disco?

CAPITOLO 3

Istruzioni di assegnazione e di controllo

La particolare versatilità del TI BASIC permette di prendere delle decisioni logiche ed eseguire un adatto insieme di ordini, subordinati al risultato di quelle decisioni. Vediamo meglio questo importante argomento.

3.1 **LET**

Questa istruzione si usa per assegnare ad una variabile un valore numerico o di stringa (alfanumerico), ed è un istruzione che non si può usare se non all'interno di un programma.

L'enunciato LET è composto da: numero di linea, comando LET, termine di assegnazione. Il termine di assegnazione è a sua volta costituito da una variabile, un segno di = e da un numero od una formula. Vediamo qualche esempio:

10 LET X = 12.5	(assegna ad x il valore 12.5)
20 LET C1 = F3	(assegna a C1 il valore contenuto nella variabile F3)
120 LET A = 3.14 * R ∧ 2	(assegna ad A il valore...)
240 LET M$ = "NOME"	(assegna alla variabile di stringa M$ il valore: NOME).

Come si può notare dagli esempi, non si può assegnare un valore numerico ad una variabile a stringa e viceversa. Se una stringa, poi, compare in un enunciato LET, la sua assegnazione deve sempre essere racchiusa tra virgolette (vedi esempio linea 240).

Nel TI BASIC, come nella maggior parte delle versioni BASIC, l'enunciato LET è comunque opzionale. Ciò significa che, per esempio, la linea 10 del precedente esempio può essere così riscritta:
10 X = 12.5

Concludiamo questo paragrafo facendo due osservazioni. Dovendo ogni formula rappresentare un valore numerico, è necessario che al momento dell'esecuzione il calcolatore sappia qual'è il valore corrispondente a ciascuna

variabile che compare nella formula, altrimenti si possono creare degli errori di tipo logico.

Vediamo un esempio:

```
10 A=5
20 B=3
30 C=7
40 D=A+B-C
50 PRINT D
60 END
```

Facciamo girare il programma ed osserviamo che il calcolatore stampa 1. Proviamo ora a far girare quest'altro programma:

```
10 A=5
20 B=3
30 D=A+B-C
40 C=7
50 PRINT D
60 END
```

Il calcolatore stampa 8 in quanto quando interpreta la linea 30 non conosce ancora il valore di C e quindi lo considera = a zero (questo è un errore di tipo logico).

Infine teniamo a precisare, anche se può sembrare superfluo, che non è permesso dare ad una variabile il nome di una parola riservata (vedi elenco parole riservate sul manuale del TI 99/4A).

3.2 GOTO

Di regola, gli enuncati di un programma BASIC vengono eseguiti nell'ordine indicato dal numero di linea, uno dopo l'altro; talvolta, però, è necessario saltare ad un'altra parte del programma, alterando la normale sequenza di esecuzione. Questo si ottiene con l'enunciato GOTO N («vai alla linea N») dove N deve essere il numero di una linea esistente nel programma.

Il numero di linea deve essere indicato sempre direttamente cioè non può essere espresso mediante una variabile anche se precedentemente definita.

Questa è un'istruzione di salto incondizionato.

Il GOTO è costituito da un mumero d'enunciato seguito dall'ordine GOTO e dal numero di linea su cui bisogna trasferire il controllo. Es.:

```
10 GOTO 100
```

Il calcolatore trasferisce il controllo al numero di linea 100, appena sequenzialmente arriva al numero di linea 10. Se il numero di linea indicato dal GOTO non esiste, il calcolatore si blocca e stampa: BAD LINE NUMBER.

3.3 **ON-GOTO**

Tra ON e GOTO va inserita una variabile o una espressione numerica e dopo il GOTO una serie di numeri di linea interspaziati da una virgola. Questo enunciato serve per trasferire il controllo in più punti del programma in dipendenza del valore assunto dalla variabile o dall'espressione numerica. Se la variabile assume il valore 1 il controllo passa alla prima linea della serie, se vale 2 alla seconda e così via.

Vediamo un esempio:

```
30 ON K GOTO 15,40,25,40,60
```

Se K ha valore 1 il calcolatore va ad eseguire la linea 15; se vale 2 o 4 la linea 40; se vale 3 la linea 25 e se vale 5 quella 60.

Nel TI BASIC è previsto che quando la macchina calcola il valore dell'espressione numerica, lo arrotondi ad un intero, trascurando la parte decimale. Es.: 5.8 è arrotondato a 5.

Se uno dei numeri di linea che seguono il GOTO non esiste nel programma, il calcolatore si arresta dopo aver stampato un messaggio del tipo: BAD LINE NUMBER.

Se invece il calcolatore dell'espressione numerica, arrotondato, è minore di 1 o maggiore del numero delle linee specificate, il programma si arresta e compare un messaggio: BAD VALUE IN N (dove N indica il numero di linea dove si trova l'enunciato ON-GOTO).

3.4 **IF-THEN e IF-THEN-ELSE**

Con questo enunciato si conduce un'operazione di salto condizionato. Esso è costituito dalle parole chiave IF (se) e THEN (allora), separate da una relazione matematica e seguite da un numero di linea. Se la relazione è soddisfatta, il controllo viene trasferito al numero di linea che segue la parola chiave THEN; altrimenti viene eseguito sequenzialmente l'enunciato successivo.

Attenzione! Per le operazioni di salto condizionato dobbiamo avere un modo per esprimere delle condizioni di eguaglianza o di diseguaglianza. A tale scopo, in BASIC, vengono introdotti gli operatori relazionabili che sono:

= uguale a
<> non uguale a
< minore di
<= minore o uguale a
> maggiore di
>= maggiore o uguale a

Gli operatori relazionali si usano per collegare grandezze numeriche (numeri, variabili o espressioni matematiche) formando così delle condizioni che possono essere o non essere soddisfatte.

Facciamo ora un applicazione dell'IF THEN:

```
15 . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . .

50 IF I >= 100 THEN 80
55 I=I+1
60 GOTO 15


80 . . . . . . . . .
```

Il modo con cui il programma viene eseguito dipende dalla relazione I >= 100 contenuta nell'enunciato IF-THEN. Se essa è soddisfatta (cioè se il valore di I è maggiore o uguale a 100) subito dopo viene eseguito l'enunciato numero 80: se non è soddisfatta (cioè se il valore di I è minore di 100), allora verrà eseguito l'enunciato numero 55.

Il programma appena scritto è un esempio di quello che, nella letteratura «software», viene indicato con il termine «loop» (ciclo). Che cos'è un loop? Molto

spesso, nei programmi, si rende necessario ripetere una certa procedura (parte di programma) per un certo numero di volte. Questo numero di volte non sempre è specificato. Il loop è la parte di programma che viene ripetuta e può contenere un'operazione di salto condizionato a causa della quale il ciclo viene interrotto. Nell'esempio precedente il loop è la parte di programma compresa tra i numeri di linea 15 e 60. Il controllo è effettuato dalla linea numero 50.

In TI BASIC, esiste un'opzione per il comando IF-THEN che permette di trasferire il controllo del programma in due punti distinti, in dipendenza del risultato del confronto. Il comando è così strutturato:

IF...THEN...ELSE... (se...allora...altrimenti...)

Questo comando si differenzia dalla già nota istruzione IF-THEN per il fatto che mentre IF-THEN trasferisce il controllo ad una linea prefissata, se viene verificata la proposizione e fa proseguire nel programma se non lo è, l'IF-THEN-ELSE in entrambi i casi manda a due linee stabilite.

Questa istruzione può essere utile in un caso come il seguente. Supponiamo di dovere prendere tre decisioni distinte in dipendenza del segno della variabile A. Facendo uso soltanto dell'IF-THEN, si è costretti ad operare così:

```
100 IF A>0 THEN 200
110 IF A<0 THEN 300
120 IF A=0 THEN 400
```

Usando invece anche l'IF-THEN-ELSE si ha:

```
100 IF A>0 THEN 200
110 IF A<0 THEN 300 ELSE 400
```

In questo modo si è usato un'istruzione in meno.

3.5 **FOR-TO**

Abbiamo già visto che in BASIC si può costruire un loop usando gli enunciati IF-THEN e GOTO e questo è necessario quando non si sa in anticipo

il numero di ripetizioni del loop. Spesso, però, questo numero di ripetizioni è noto sin dall'inizio e allora è molto più comodo servirsi degli enunciati FOR-TO e NEXT.

Il primo di questi specifica quante volte il loop sarà eseguito, e deve essere sempre il primo enunciato nel corpo del loop. Tra le due parole chiave FOR e TO, va inserita un assegnazione. La variabile numerica che compare in questa assegnazione è anche detta «running variable» (variabile corrente) ed il suo valore cambia ogni volta che il loop viene eseguito. Vediamo qui un tipico enunciato FOR TO:

```
50 FOR I=1 TO 10
```

In questo caso il ciclo è ripetuto per un numero di 10 volte. Durante queste ripetizioni la variabile corrente I assume tutti i valori tra 1 e 10 compresi gli estremi.

Se si vuole, è possibile incrementare la variabile corrente con un valore diverso da 1 (sempre intero, però). Tale incremento va specificato dopo il valore finale e con l'ausilio della parola chiave STEP.

Per chiarire questo punto vediamo ora alcuni esempi:

```
30 FOR X=-1.5 TO 2.7 STEP 0.1
```

In questo caso, la variabile corrente, incrementerà il valore iniziale -1.5 di 0.1 fino a raggiunger il valore finale 2.7.

```
15 FOR I=N TO 0 STEP -1
```

Questo esempio, invece, mostra come il valore iniziale della variabile corrente possa anche essere una variabile numerica (naturalmente il calcolatore al momento della linea 15 deve sapere il valore di N).

Seguono altri possibili esempi:

```
55 FOR K=N1 TO N2 STEP N3
80 FOR AA=F/2 TO (B+C)^2 STEP K+1
```

3.6 **NEXT**

Così come un loop inizia sempre con un istruzione FOR-TO, così esso deve terminare con l'enunciato NEXT. Il NEXT è costituito da un numero di linea, dal comando NEXT e dalla variabile corrente del ciclo a cui si riferisce: la stessa che compare nell'istruzione FOR-TO.

Vediamo ora la struttura essenziale di un loop:

```
50 FOR I=1 TO 10

90 NEXT I
```

Il loop è quindi costituito, in questo caso, da tutti gli enunciati compresi tra la linea 50 e la linea 90 e sarà ripetuto 10 volte.

Ci preme a questo punto sottolineare alcune regole che bisogna tener presenti quando si costruisce un loop con il FOR-TO-NEXT.

1. La variabile corrente può anche apparire in un enunciato all'interno del loop, ma il suo valore deve restare inalterato. Cioè:

```
10 FOR K=1 TO 10
20 A=(K+1)^2                    VALIDO
30 PRINT A,K
40 NEXT K
```

```
10 FOR K=1 TO 10
20 K=(K+1)^2                    NON VALIDO
30 PRINT K
40 NEXT K
```

2. Se il valore iniziale e quello finale della variabile corrente sono uguali ed il passo assegnato è diverso da zero, il loop sarà eseguito una sola volta.
3. Il loop non verrà mai eseguito se sussiste una delle tre condizioni particolari seguenti:
   a) il valore iniziale e finale della variabile corrente sono uguali e lo STEP è zero;
   b) il valore finale della variabile corrente è maggiore di quello iniziale e lo

STEP è positivo;

c il valore finale della variabile corrente è maggiore di quello iniziale e lo STEP è negativo.

4. Il controllo può trasferirsi dall'interno di un loop verso l'esterno ma non viceversa.

3.7. END

L'enunciato END indica la fine fisica di un programma BASIC ed è costituito da un numero di linea seguito dal comando END.

Questo enunciato deve essere sempre l'ultimo del programma e deve sempre essere contraddistinto dal numero più alto di linea.

Nel TI BASIC l'istruzione END non è sempre necessaria.

3.8 STOP

Questa istruzione serve ad interrompere il calcolo in un punto qualsiasi del programma. Essa equivale, logicamente parlando, ad un GOTO che trasferisce il controllo ad un enunciato END, e consiste in un numero di linea seguito dalla parola STOP.

È importante capire la differenza tra STOP ed END: il primo può apparire ovunque e più volte in un programma ma non alla fine fisica del suddetto; il secondo, invece, può apparire soltanto alla fine del programma e quindi, nel medesimo, si può usare una sola volta.

ESERCIZI RISOLTI (CAP. 3)

1) Eseguire le seguenti assegnazioni:
ad A 3,71; a B$ la stringa MARZO 1983; ad I l'espressione I+1 e a C$ la variabile S$.

Soluzione:

```
10 LET A=3.71
20 LET B$="MARZO 1983"
30 LET I=I+1
40 LET C$=S$
```

2) Quando un espressione algebrica è particolarmente complicata e si desidera assegnarla ad una variabile, è conveniente spezzare l'assegnazione in assegnazioni multiple. Mostrare come ciò sia possibile nel seguente caso:

$t = (3xy/x + a + b^2)^{m\ n}$

Soluzione:

```
10 LET T1= (3*X*Y)/X
20 LET T2= A+B∧2
30 LET T= (T1+T2)∧(M-N)
```

3) Verificare la presenza di errori sintattici nelle seguenti istruzioni e motivarli.

```
a) 10 GOTO 140
b) 20 GOTO M
c) 30 IF K∧2>= 100 THEN 50
d) 40 THEN 300 IF A=B
e) 20 IF X1<= 71 THEN J=J+1
f) 100 IF X= A GOTO 10
g) 55 ON K GOTO A,B,C
h) 70 GOR I=1 TO M STEP I
```

Soluzione:
a) corretta;

b) non corretta in quanto il numero di linea a cui viene trasferito il controllo deve essere una costante numerica e non una variabile;
c) corretta;
d) non corretta in quanto l'istruzione deve cominciare con la parola chiave IF;
e) non corretta in quanto in TI BASIC è necessario che il THEN sia seguito da un numero di linea e non da un'altra istruzione (ciò è invece possibile in EXTENDED BASIC);
f) non corretta perché in TI BASIC è necessario il THEN e non il GOTO;
g) non corretta perché i numeri di linea a cui si trasferisce il controllo devono essere costanti numeriche e non variabili (analogamente all'esercizio b);
h) non corretta perchè la variabile corrente del FOR non può comparire come argomento dello STEP.

4) Identificare se le seguenti strutture IF-THEN... GOTO sono corrette:

a)
```
50 .............
70 IF N=M THEN 200
90 LET N=N+1

120 GOTO 50

200 ............
```

b)
```
10 C=1
20 IF C<=0 THEN 100
30 C=C+1

100 ............
```

Soluzione:
a) corretta;
b) non corretta dal punto di vista logico. Infatti la variabile C non sarà mai negativa e quindi il ciclo continuerà all'infinito (o meglio indefinitamente).

5) Verificare l'esattezza delle seguenti strutture FOR-TO...NEXT:

```
a) 30 FOR I=1 TO 10 STEP 2
   40 ..............

   50 NEXT I

b) 50 FOR N=1 TO 20
   ................

   70 ..............


   100 NEXT N
   110 IF A=B THEN 70
```

Soluzione:
a) corretta;
b) non corretta in quanto non è possibile trasferire il controllo del programma all'interno di un loop FOR-TO-NEXT.

ESERCIZI PROPOSTI (CAP.3)

1) È possibile evitare l'enunciato LET come nel seguente esempio?

10 A=5 anzichè 10 LET A=5

2) Individuare quale dei seguenti enunciati è corretto e quale no. Se scorretto specificarne il motivo.

```
10 IF A=BS THEN 100
20 ON (X-12) GOTO 100,200,300
15 LET A="MARIO ROSSI
40 IF A=B1 THEN LET X=1
```

3) Consideriamo la seguente istruzione:

```
30 ON (K-B) GOTO 100,200,500,100
```

discutere che cosa accade nei seguenti casi:

a) K=1; B=2 b) K=.5; B=3 c) K=7; B=4 d) K=-1; B=1

4) In TI BASIC, è possibile che la variabile corrente di un loop FOR-TO...NEXT assuma valori decimali o anche negativi?

5) Spiegare sommariamente la funzione dei tre seguenti enunciati: LET, STOP, END.

CAPITOLO 4

Istruzioni di ingresso e uscita

Quando si parla di istruzioni d'ingresso si intende tutta una serie di comandi che servono ad introdurre in memoria dati che vengono usati nel corso del programma.

Invece, per istruzioni di uscita si intende quella serie di comandi che bisogna utilizzare per ottenere dal computer i risultati richiesti. È ovvio che in base alla natura dei dati (o dei risultati) da trattare, ci sono opportune istruzioni da usare. Vediamole insieme!

4.1 **INPUT**

È un'istruzione di ingresso.

Questo enunciato si usa per introdurre dati, numerici o a stringa, nel calcolatore durante l'esecuzione di un programma.

Esso è costituito da un numero di linea, dal comando INPUT e da una lista di variabili, che possono essere numeriche o a stringa, sempre interspaziate da virgole.

Vediamo alcuni esempi:

```
15 INPUT A,B,C
110 INPUT N$, M$, XO, F5
```

Se durante l'esecuzione di un programma il calcolatore incontra un enunciato INPUT, si arresta e stampa sul monitor un punto interrogativo; il programmatore a questo punto deve fornire i dati richiesti, tramite tastiera, e quindi battere ENTER. In tal modo i dati passano al calcolatore e ricomincia l'elaborazione del programma.

All'atto di inserire i dati bisogna osservare queste regole:

1. I dati debbono corrispondere in numero ed in tipo alle variabili elencate (per esempio per variabili numeriche, numeri, e per variabili alfanumeriche, stringhe). Dati non corrispondenti vengono ignorati.
2. I dati vanno *assolutamente* separati da virgole.
3. I dati debbono essere costituiti da numeri o da stringhe e non da formule.

4. I dati alfanumerici possono anche essere racchiusi tra virgolette (ciò è facoltativo): questa operazione si rende necessaria quando qualcuno tra i caratteri di cui è composta la stringa è una virgola o uno spazio bianco.

Concludiamo con un esempio che, a nostro avviso, può essere molto utile per chiarire tutto ciò che abbiamo detto in precedenza.

```
100 ..........
..............

230 INPUT X,Y,C$
240 ..........
..............
```

Appena il calcolatore arriva alla linea 230, l'esecuzione si interrompe e sul monitor appare un punto interrogativo (e per alcune macchine è prevista l'emissione di un beep). Quando l'utente vede il suddetto punto interrogativo deve fornire i dati richiesti.

Supponiamo, a questo punto, che i dati corretti da inserire siano: X = 5; $Y = 1.2x10^{-3}$; C$ = NOVEMBRE 27, 1937
La riga dei dati sarà allora:

```
? 5, + 1.2 E-3, "NOVEMBRE 27, 1937"
```

Una volta battuti i dati, si preme ENTER ed il tutto viene trasmesso al calcolatore. Il resto del programma viene eseguito nel modo consueto.

L'enunciato INPUT è molto utile nei programmi elementari che non richiedono grandi quantità di dati in ingresso. Si tratta però di una procedura che richiede tempo ed i dati così introdotti non possono venire immagazzinati per successivo uso. Vedremo, in seguito, come invece ciò sia possibile con le istruzioni READ e DATA.

4.2 **READ e DATA**

In molti programmi BASIC è necessario fornire, in ingresso al calcolatore, una grande quantità di dati (ad esempio per elaborazioni gestionali) e spesso questi dati vengono utilizzati più volte. Per far ciò l'istruzione INPUT non è più adeguata e, al suo posto, si usano gli enunciati READ e DATA.

Parliamo di queste due istruzioni insieme, poiché non ha senso usare l'una senza l'altra.

L'enunciato READ specifica le variabili alle quali verranno assegnati i valori contenuti nel DATA. Esso consiste in un numero di linea seguito dall'ordine READ e da una lista di variabili. La lista può contenere sia variabili numeri-

che che alfanumeriche. Se il numero delle variabili del READ è superiore ad uno, esse vanno spaziate con la virgola.

Scopo dell'enunciato DATA è di assegnare i valori appropriati alle variabili elencate nell'enunciato READ. Esso è costituito da un numero di linea seguito dal comando DATA e da un insieme di numeri e/o di stringhe separati da virgole.

Ogni numero (o stringa) nel DATA deve corrispondere rispettivamente ad una variabile dello stesso tipo nel READ, secondo lo stesso ordine. Ad esempio:

```
30 READ K,N$,Z,M$

70 DATA 12, DICIASSETTE, -5, MARIO
```

In tal modo il numero 12 viene assegnato alla variabile K, la stringa «diciassette» alla variabile alfanumerica N$, -5 alla variabile Z e «Mario» alla variabile alfanumerica M$.

Vediamo ora in sintesi le regole che dobbiamo rispettare quando costruiamo un blocco dati.

1. I dati nel DATA debbono corrispondere in ordine e tipo alle variabili elencate nell'enunciato READ; gli elementi del DATA devono essere tanti quanti quelli nel READ; se il numero dei dati è superiore a quello delle variabili, gli eccedenti verranno ignorati.
2. I dati del DATA devono sempre essere separati da virgole tutti tranne l'ultimo.
3. I dati debbono essere numeri o stringhe e non variabili o formule.
4. Le stringhe che contengono delle virgole o degli spazi bianchi devono essere sempre racchiuse tra virgolette.
   Esempio:

```
10 READ A,B
. . . . . . . . . . .
60 DATA 123, "MARE CALMO"
```

Osserviamo che i dati specificati con gli enunciati READ e DATA sono parte integrante di un programma a differenza di quelli inseriti con l'enunciato INPUT.

N.B. -Due virgole consecutive nel DATA, assegnano alla corrispondente variabile del READ il carattere nullo.

4.3 **PRINT**

Questo enunciato è di fondamentale importanza poiché permette di stampare sul monitor qualsiasi cosa ed in particolare i risultati di un'elaborazione. Esso è costituito da un numero di linea dal comando PRINT e da una lista di dati da stampare che possono essere numeri, formule o stringhe. Gli elementi della suddetta lista devono essere separati da virgole, da punti e virgola o due punti.

Facciamo un esempio:

```
10 INPUT A,B
20 C=A+B
30 PRINT C
40 STOP
50 END
```

Come ben sapete (almeno speriamo!) alla linea 10 il calcolatore si ferma e attende due dati. Una volta ottenuti da consolle, il programma va avanti e calcola la somma A+B. Giunto alla linea 30 trova il comando di stampa per la variabile C (che naturalmente è uguale ad A più B), e sul monitor appare il numero C.

Questo è un esempio di stampa di una variabile numerica. Se ora vogliamo stampare una variabile alfanumerica o una frase, dobbiamo operare così:

```
100 PRINT "COME TI CHIAMI?"
110 INPUT A$
120 PRINT "CARO"; A$; "SONO FELICE DI FARE LA TUA CONOSCENZA"
130 STOP
140 END
```

Cerchiamo ora di spiegare questo programma. Alla linea 100 il calcolatore stampa un messaggio (o costante alfanumerica) che per essere accettato va sempe racchiuso tra virgolette. Invece, come si nota dalla linea 120, una variabile alfanumerica (A$) non deve essere racchiusa tra virgolette come le costanti alfanumeriche.

Vediamo ora qual'è la differenza tra i tre elementi separatori. Esempio:

```
100 INPUT A,B,C
```

```
110 D=A+B
120 E=(A-C)/B
130 F=A/C
140 PRINT D,E,F
```

Supponendo che i risultati di questa elaborazione siano:

```
D=3    E=8    F=12.5
```

la stampa sullo schermo sarà del tipo:

```
3          8
12.5
```

Cioè ogni riga viene divisa in due «campi» uguali, quindi su ogni riga verrà stampato al massimo il valore di due variabili.

Supponiamo ora di sostituire la linea 140 con la seguente:

```
140 PRINT D;E;F
```

si otterrà allora una stampa del tipo:

```
3  8  12.5
```

Cioè vengono lasciati due spazi tra un numero e il successivo.

Supponiamo ora di sostituire la linea 140 con la seguente:

```
140 PRINT D:E:F
```

si otterrà allora una stampa del tipo:

```
3
8
12.5
```

A questo punto vogliamo parlare di una particolarità dell'enunciato INPUT. Abbiamo preferito trattarlo ora perché ha qualcosa a che vedere con la stampa. Per questo enunciato è prevista la possibilità di stampare, oltre al punto interrogativo (che appare automaticamente ogni qualvolta si usa l'INPUT), un messaggio opzionale. Ad esempio:

```
10. INPUT "DAMMI UN NUMERO":N
```

sul monitor compare allora la frase:

```
DAMMI UN NUMERO?
```

È necessario, dopo aver scritto il messaggio dividerlo dalla variabile con l'elemento separatore:

Questa particolarità diventa essenziale quando in un programma sono presenti diversi enunciati INPUT. Infatti, in questo caso, è comodo per il programmatore che ad ognuno di essi sia associato un messaggio che aiuta a ricordare di quale INPUT si tratta, che tipo e quanti dati debba inserire.

Concludiamo ora con la funzione TAB. Questa funzione ha lo stesso effetto del tasto tabulatore di una macchina da scrivere e serve appunto per posizionare la stampa.

Ad esempio se scriviamo le seguenti linee di programma:

```
100 A=100
110 B=200
120 PRINT A; TAB(15)B
```

avremo sul monitor:

```
100           200
```

Ossia la macchina stampa la variabile A e quindi si sposta sulla quindicesima colonna per stampare B. L'argomento della TAB indica la colonna su cui va stampata la variabile che segue.

4.4 RESTORE

Parlando del READ e del DATA abbiamo visto che i dati vengono assegnati a partire dal primo dato nel primo DATA che s'incontra nel programma e via via in succèssione fino all'ultimo dato dell'ultimo DATA (questo naturalmente se il numero delle variabili nel READ è uguale alla somma degli elementi di tutti i DATA presenti nel programma).

Nel caso si voglia far rileggere gli stessi dati ad un nuovo gruppo di variabili, dobbiamo avvertire di ciò la macchina con l'enúnciato RESTORE, che va posto immediatamente prima del READ che si riferisce a queste nuove variabili.

Questa istruzione può essere anche seguita da un numero di linea corrispondente ad un DATA. In questo caso la rilettura avviene a partire dal primo dato relativo al DATA indicato dal numero di linea. È evidente che usare il numero di linea ha senso solo se nel programma sono presenti più DATA. Esempio:

```
100 READ A,B,C
 . . . . . . . . . . . . .

160 RESTORE
180 READ X,Y,Z,T,U
 . . . . . . . . . . . . .

300 DATA 1,2,3
310 DATA -7,-3,-1,7,6
320 END
```

L'istruzione 100 assegna ad A,B e C rispettivamente i valori 1,2 e 3; l'istruzione 160 fa si che alle variabili X,Y,Z,T,U siano assegnati rispettivamente i valori 1,2,3,-7,-3; se non ci fosse stata questa istruzione le precedenti variabili avrebbero assunto i valori -7,-3,-1,7,6.

Le applicazioni pratiche di questa istruzione forse ora non sono molto evidenti, ma si capirà meglio la sua utilità quando si tratteranno gli array mono e bi-dimensionali.

ESERCIZI RISOLTI (CAP.4)

1) Qual'è l'istruzione del programma che dobbiamo usare per stampare sul video il contenuto delle variabili A,B e C? E quando (cioè in che punto del programma) deve essere inserita?

Soluzione: L'istruzione da usare è PRINT e va inserita alla fine della elaborazione (ossia quando in A,B e C ci sono i valori finali). Il comando è:

```
100 PRINT A,B,C
```

2) Qual'è un possibile modo per stampare sul video i valori di tre variabili con appropriata identificazione?

Soluzione: Supponiamo che le variabili siano X,Y e Z. Allora si avrà:

```
100 PRINT "X="; X:"Y=";Y:"Z=";Z
```

3) Scrivere un enunciato INPUT che chieda un numero intero e ponga tale valore in una variabile di nome N.

Soluzione:

```
100 INPUT "DAMMI UN NUMERO INTERO":N
```

4) Usare gli enunciati READ e DATA per assegnare alle variabili A,B,C,$,X,Y$ e Z i valori: 312; - $51{,}7 \times 10^{-11}$; MARZO;
31,4; MAMMA; $-7{,}121 \times 10^{18}$

Soluzione:

```
100 READ A,B,C$,X,Y$,Z
200 DATA 312,-51.7E-11, MARZO, 31.4
210 DATA MAMMA, -7.121E18
```

5) Data la funzione $y=3x^2$, scrivere un programma che stampi una tabella di valori di y al variare di x nell'intervallo (-2,2) con un incremento qualsiasi (si desidera che la stampa dei risultati sia del tipo per X=1 Y=3)

Soluzione: È facile verificare che un programma del genere può essere il seguente:

```
100 FOR X=-2 TO 2 STEP 0.2
110 Y=3*X∧2
120 PRINT "PER X=";X,"Y=";Y::
130 NEXT X
```

ESERCIZI PROPOSTI (CAP.4)

1) È obbligatorio inserire il messaggio (tra virgolette) dopo un enunciato INPUT?

b) In caso affermativo, qual'è l'elemento separatore tra tale messaggio e la (o le) variabile?

2) A che cosa serve la funzione TAB?

b) In quale enunciato si usa?

3) È necessario che per ogni enunciato READ ci sia un enunciato DATA?

b) È importante l'ordine dei dati nel comando DATA?

4) Spiegare lo scopo dell'enunciato RESTORE.

5) Fare un programma che calcoli la media aritmetica tra 10 numeri inseriti con gli enunciati READ e DATA.

CAPITOLO 5

Funzione di libreria

Le funzioni di libreria del TI BASIC forniscono un modo rapido e facile per calcolare diverse funzioni matematiche e per eseguire delle operazioni logiche. In realtà sono sottoprogrammi scritti in linguaggio macchina e residenti permanentemente nel sistema operativo che si richiamano mediante opportune chiavi.

Una volta richiamata una funzione di libreria, l'operazione desiderata viene fatta automaticamente senza bisogno di un esplicito e dettagliato programma.

Vediamo ora quali sono le funzioni di libreria del TI 99/4A (qui vengono illustrate come singole linee di programma in cui si assegna alla variabile y la funzione di libreria):

10 Y=ABS(X)	trasferisce in Y il valore assoluto di X.
10 Y=ATN(X)	trasferisce in Y l'arco la cui tangente è X.
10 Y=COS(X)	trasferisce in Y il coseno di X.
10 Y=EXP(X)	trasferisce in Y esponenziale di X.
10 Y=INT(X)	trasferisce in Y il massimo intero che non superi algebricamente X.
10 Y=LOG(X)	trasferisce in Y il logaritmo naturale di X.
10 Y=SGN(X)	trasferisce in Y+1 se X è maggiore di zero, -1 se X è minore di zero, 0 se X è uguale a zero.
10 Y=SIN(X)	trasferisce in Y il seno di X.
10 Y=SQR(X)	trasferisce in Y la radice quadrata di X.
10 Y=TAN(X)	trasferisce in Y la tangente di X.

L'uso di queste funzioni è di per se evidente qualora se ne conosca il signifi-

cato matematico. In caso contrario il lettore può anche saltare l'argomento.

Osserviamo che alcune di esse (LOG, SQR) richiedono un argomento positivo; nel caso in cui si assegni invece un argomento negativo il sistema si ferma e stampa sul monitor un messaggio del tipo:
"BAD ARGUMENT IN N"
dove N è la linea di programma dove si fa riferimento ad una delle suddette funzioni con argomento negativo.

La funzione INT richiede qualche precisazione. Se l'argomento di questa funzione ha valore positivo, esso viene troncato (scompare cioè la parte decimale del numero) e viene così generato un intero positivo di valore minore di quello dell'argomento. Se invece l'argomento è negativo, la funzione INT fornisce un intero negativo di valore assoluto maggiore dell'argomento. Vediamo un esempio:

```
10 LET Y= INT(X)
se X=12.9     allora Y=12
se X=-4.2     allora Y= 5
```

5.1 **RND**

In TI BASIC si può generare molto facilmente un numero casuale per mezzo della funzione di libreria RND. Ogni volta che si introduce in un programma questa funzione, viene generato a caso un numero compreso tra zero e uno.

Per questa funzione non è richiesto argomento. Se ad esempio vogliamo generare un numero casuale compreso tra 1 e 10, dovremo usare la seguente formula:

```
100 X=1+INT (RND* 9)
```

In effetti il numero così generato sarà anche intero.

In generale se si vuole estrarre un numero compreso tra A e B si deve usare la formula seguente:

```
X=A+(B-A)* RND
```

5.2 **RANDOMIZE**

Abbiamo visto come si generano numeri casuali con la funzione RND. Supponiamo però di avere un programma che genera una serie di numeri casuali con la funzione RND e di dover far girare questo programma, per esigenze particolari, più volte consecutivamente. Ogni volta che noi diamo il RUN si otterrà *sempre la stessa* sequenza di numeri.

Per ovviare a ciò esiste un'istruzione BASIC chiamata RANDOMIZE che, posta prima dell'istruzione RND, fa si che le successive sequenze siano diverse tra di loro ogni volta che diamo il comando RUN.

A titolo esemplificativo proviamo a far girare il seguente programma più volte per verificare quanto detto:

```
10 FOR I=1 TO 7
20 A=INT (9*  RND)+1
30 PRINT A
40 NEXT I
50 STOP
60 END
```

Come si può facilmente osservare dal monitor si tratta di sequenze tutte uguali.

Proviamo invece ora a far girare più volte lo stesso programma aggiungendo la linea:

```
15 RANDOMIZE
```

Osserveremo, ora, che le successive sequenze sono fra loro essenzialmente diverse.

ESERCIZI RISOLTI (CAP. 5)

1) Generare e stampare una sequenza di 10 numeri casuali interi e compresi tra 1 e 90.

Soluzione: Se vogliamo che ogni volta che diamo il RUN, la sequenza sia sempre diversa, è necessario iniziare il programma con l'enunciato RANDOMIZE.

```
100 RANDOMIZE
110 FOR J=1 TO 10
120 A=INT (RND*90)+1
130 PRINT A
140 NEXT J
```

2) Scrivere un programma che assegni alla variabile y il valore $(|\cos x - \sin x|)^{0.5}$ per x che va da 10 a 100 di 10 in 10 e che stampi di volta in volta il valore di y.

Soluzione:

```
10 FOR X=10 TO 100 STEP 10
20 Y=SQR (ABS(COS(X)-SIN(X)))
30 PRINT Y
40 NEXT X
```

3) Fare un programma che verifichi se un dato numero è pari o dispari.

Soluzione: Sappiamo che la funzione INT tronca la parte decimale di un numero. Se però il numero è pari, la divisione per 2 lo rende sempre intero e quindi si verifica che N/2=INT(N/2) (dove con N si è indicato il numero incognito). Vediamo il programma:

```
100 INPUT "DAMMI UN NUMERO":N
110 IF N/2=INT (N/2) THEN 140
120 PRINT "IL NUMERO";N;"È DISPARI"
130 STOP
140 PRINT "IL NUMERO";N;"È PARI"
150 STOP
```

4) Determinare con l'ausilio della funzione SGN (X) il segno della variabile numerica X in modo tale che se X< 0 mandare il programma all'enunciato 100, se X=0 al 200 e se X>0 al 150.

Soluzione:

```
100 ON SGN(X)+2 GOTO 100,200,150
```

5) Che cosa s'intende per numeri pseudo-casuali? In che cosa differiscono da quelli realmente casuali?

Soluzione: Con questo termine si indicano numeri casuali generati con una ben determinata procedura. Essi hanno comunque le stesse proprietà statistiche dei numeri realmente casuali.

Quindi qualunque tipo di studio che coinvolge numeri casuali, può, in linea di principio e nei limiti della macchina, essere affrontato con il TI 99/4A.

ESERCIZI PROPOSTI (CAP. 5)

1) Che cosa s'intende per argomento? Tutte le funzioni di libreria hanno bisogno di un argomento?

2) Che cosa succede se ad una funzione di libreria che richiede un argomento positivo ne viene fornito uno negativo?

3) L'argomento delle funzioni di libreria trigonometiche, deve essere dato in gradi oppure in radianti?

4) Che cosa stampa il seguente programma?

```
100 A=2.16
110 B=3.41
120 C=INT(A)
130 D=INT(B)
140 PRINT C,D
```

5) Fare un programma che generi, ogni volta che il calcolatore lo esegue, 2 numeri casuali diversi tra loro e compresi nell'intervallo (3,20).

CAPITOLO 6

Funzioni di stringa

Abbiamo già visto in precedenza qual'è il signficato del termine stringa o costante alfanumerica. Nei paragrafi che seguono vengono introdotte alcune funzioni di libreria molto particolari che prendono il nome di funzioni di stringa.

6.1 **Il codice A.S.C.I.I.**

Prima d'illustrare le suddette funzioni è opportuno parlare della codifica dei dati. Quando nel calcolatore viene immessa una stringa (cioè un insieme di caratteri qualsiasi), i caratteri che la compongono vengono immagazzinati come una sequenza codificata di numeri. Ogni carattere (cifra, lettera o carattere speciale), infatti, viene rappresentato all'interno della macchina da un suo unico e particolare numero; si stabilisce così una corrispondenza biunivoca tra l'insieme dei caratteri ed un opportuno insieme di numeri. Non vi è un solo modo per attuare questa corrispondenza! Esistono, infatti, diversi schemi di codifica numerica: quello adottato per il TI 99/4A è il codice ASCII che, d'altra parte, è quello usato nella maggior parte dei calcolatori. Esso stabilisce una corrispondenza tra i caratteri ed i numeri interi positivi compresi tra 32 e 127 estremi inclusi ed è riportato qui di seguito.

Carattere	Codice ASCII	Carattere	Codice ASCII
cursore	30	O	79
margine	31	P	80
spazio	32	Q	81
!	33	R	82
"	34	S	83
#	35	T	84
$	36	U	85
%	37	V	86
&	38	W	87
'	39	X	88
(	40	Y	89
)	41	Z	90

Carattere	Codice ASCII	Carattere	Codice ASCII
*	42	[	91
+	43	\	92
,	44	]	93
-	45	^	94
.	46	-	95
/	47	`	96
0	48	a	97
1	49	b	98
2	50	c	99
3	51	d	100
4	52	e	101
5	53	f	102
6	54	g	103
7	55	h	104
8	56	i	105
9	57	j	106
:	58	k	107
;	59	l	108
<	60	m	109
=	61	n	110
>	62	o	111
?	63	p	112
@	64	q	113
A	65	r	114
B	66	s	115
C	67	t	116
D	68	u	117
E	69	v	118
F	70	w	119
G	71	x	120
H	72	y	121
I	73	z	122
J	74	{	123
K	75	}	124
L	76		125
M	77	~	126
N	78	DEL	127

È molto importante che l'utente prenda una certa confidenza con tale codice in quanto, come vedremo più avanti, viene spesso richiamato nelle funzioni di grafica.

6.2 **ASC e CHR$**

La funzione ASC converte ogni singolo carattere nel suo equivalente ASCII; se il suo argomento è una stringa di due o più caratteri, essa riconoscerà solo il primo. Ad esempio:

```
100 B=ASC ("A")
110 C=ASC (7)
120 D=ASC ("ZUZZURELLONE")
130 PRINT B;C;D
```

Eseguito questo programma, osserveremo sul monito i valori:

65 55 90

Lo scopo della funzione di libreria CHR$ è invece esattamente l'opposto di quello della funzione ASC. Con questa funzione, infatti, otteniamo il carattere alfanumerico corrispondente al numero che abbiamo dato per argomento, secondo appunto il codice ASCII. Si verifica infatti che:

```
CHR$ (65)=A
CHR$ (33)=!
CHR$ (90)=Z
```

Se il valore dell'argomento non è intero, allora viene arrotondato (per troncamento).

Queste due funzioni sono un valido ausilio nel problema del riconoscimento delle stringhe.

6.3 **LEN**

Questa funzione di libreria è usata per calcolare il numero di caratteri di cui una stringa è composta (ricordiamo che lo spazio viene considerato come un carattere). Una stringa nulla ha lunghezza nulla. Vediamo qualche esempio.

```
10 A$="GIANNI"
```

```
20 B$ = "MARIA"
30 A = LEN (A$)
40 B = LEN (B$)
50 PRINT "LA PAROLA";A$;"È COMPOSTA DA";A;"CARATTERI"
60 A$ = B$
65 A = B
70 GOTO 50
```

Il programma mostra come la funzione LEN possa essere usata per calcolare appunto il numero di caratteri di cui è composta una stringa.

6.4 **POS**

È una funzione di libreria molto particolare ed in genere non molto usata. L'enunciato è del tipo:
POS (prima stringa, seconda stringa, espressione_numerica)
Questa funzione genera un numero intero che rappresenta la posizione in cui si incontra per la prima volta la seconda stringa nella prima stringa. Il risultato dell'espressione numerica viene calcolato ed eventualmente arrotondato ad un intero n. La ricerca, così, inizia dalla posizione n-esima della stringa 1. Ad esempio:

```
100 A = POS ("CIAO", "A",1)
110 PRINT A
```

E sul monitor si ottiene il numero 3.

6.5 **SEG$**

L'enunciato generale di questa funzione è:
SEG$ (espressione di stringa, espressione numerica 1, espressione numerica 2)
Vediamo un esempio:

```
100 M$ = "BUONGIORNO A TUTTI!"
110 S$ = SEG$ (M, 14,6)
120 PRINT S$
```

Una volta eseguito questo programma il calcolatore stamperà sullo schermo la stringa TUTTI!
Infatti, il procedimento seguito è:
1) viene calcolato il valore numerico dell'espressione 1 ed eventualmente arrotondato per troncamento;
2) tale numero costituisce la posizione del carattere di partenza dell'espressione di stringa;
3) analogamente al punto 1 si opera per l'espressione numerica 2;
4) tale numero costituisce il numero di caratteri da prelevare nella stringa di partenza a cominciare da quello che si trova nella posizione indicata dal numero calcolato nell'espressione 1.

6.6 **STR$**

Questa funzione converte in stringa il numero che ne costituisce l'argomento. Ad esempio se si ha:

```
Y$=STR$(71.5)
```

in Y$ verrà trasferita la stringa "71.51".

Naturalmente sulle stringhe così ottenute non si possono fare operazioni numeriche ma solo operazioni di stringa.

L'argomento può anche essere un'espressione numerica. In tal caso ne viene calcolato il valore numerico ed è quel valore che viene convertito in stringa.

6.7 **VAL**

Questa funzione è la complementare della STR$ che abbiamo appena visto. Essa converte infatti il suo argomento, che è sempre una stringa numerica, in un numero.

L'enunciato generale è VAL (espressione di stringa). Naturalmente l'espressione di stringa deve essere sempre una corretta rappresentazione di un numero, altrimenti la conversione non ha senso. Ad esempio:

```
100 Y=VAL ("105.11")
```

in Y verrà trasferita la costante 105.11.

Se però l'argomento di VAL non è una corretta rappresentazione di un numero o se ha lunghezza zero (nel caso di stringa nulla) o se è maggiore di 254 caratteri, il programma si blocca e sul monitor compare: "BAD ARGUMENT".

ESERCIZI RISOLTI (CAP. 6)

1) Scrivere un programma che, sfruttando le istruzioni di stringa, stampi in ordine alfabetico due nomi propri di persona.

Soluzione:

```
100 INPUT "DAMMI DUE NOMI":A$,B$
110 C$=SEG$(A$,1,1)
120 D$=SEG$(B$,1,1)
130 C=ASC(C$)
140 D=ASC(D$)
150 IF C<D THEN 180
160 PRINT B$::A$
170 STOP
180 PRINT A$: :B$
```

La spiegazione di questo programma diviene chiara se si pensa al funzionamento della funzione SEG$ che, in questo caso, assegna alle variabili alfanumeriche C$ e D$ rispettivamente il primo carattere delle variabili A$ e B$. Se poi si pensa alla particolare disposizione dei caratteri alfabetici nel codice ASCII e si osservano le linee 130, 140 e 150, il gioco è fatto!

2) Usando la funzione ASC, scrivere la struttura essenziale di un programma che trasferisca il controllo del programma ad opportune linee in dipendenza del carattere battuto da tastiera.

Soluzione: Siccome il codice ASCII del carattere A è 65 è quello del carattere B è 64 e così via, allora nell'espressione K-64 ci sarà 1 se K$ è A,2 se K$ è B e così via dicendo.
Vediamo ora il programma:

```
100 INPUT K$
110 K=ASC (K$)
120 ON (K-64) GOTO 300,400...
 . . . . . . . . . . . . . .
300
 . . . . . . . . . . . . . .
400 . . . . . .
```

3) Calcolare il numero di caratteri di cui sono composte le seguenti stringhe:
A$ = CINEMA B$ = RENATO C$ = CASA

Soluzione:

```
100 A$=CINEMA
110 B$=RENATO
120 C$=CASA
130 A=LEN(A$)
140 B=LEN(B$)
150 C=LEN(C$)
```

4) Scrivere un programma che, dato un numero compreso tra 65 e 90, stampi il carattere alfabetico corrispondente secondo la codifica ASCII.

Soluzione:

```
100 INPUT "DAMMI N": N
120 IF N<65 THEN 100
130 IF N>90 THEN 100
140 A$=CHR$ (N)
150 PRINT "IL CARATTERE CHE CORRISPONDE"
160 PRINT:"AL CODICE ASCII";N;"È";A$
```

il significato delle linee di stampa 150 e 160 è di per se evidente.

5) È necessario che l'argomento del comando VAL sia un numero in «forma alfanumerica»?

Soluzione: Si lo è! In caso contrario la macchina segnalerà errore.

ESERCIZI PROPOSTI (CAP. 6)

1) Spiegare il funzionamento del comando ASC e della funzione CHR$.

2) Che cos'è e perché è stato introdotto il codice ASCII?

3) Impostare un programma che faccia uso della funzione POS ed uno che faccia uso della funzione SEG$.

4) Che cosa sono e a che servono le funzioni VAL e STR$?

5) Far girare il seguente programma e spiegarlo nei suoi dettagli:

```
10 A$="ABCDEFGHIJKLMNOPQRSTUVXYWZ"
20 FOR I=1 TO 21
30 PRINT TAB(I);
40 PRINT SEG$ (A$,I,6)
50 NEXT I
60 END
```

CAPITOLO 7

Sottoprogrammi

Con il crescere della complessità e dell'importanza dei programmi, si presentano diversi problemi immediati:

1. Gli algoritmi (per algoritmo s'intende una serie d'istruzioni in sequenza atti a risolvere un problema specifico) per risolvere problemi più complessi si fanno più difficili, quindi più faticosi da tracciare.

2. Con l'aumentare della lunghezza e della complessità dei programmi, la ricerca degli errori si fa più difficoltosa: programmi più lunghi sono per loro natura più soggetti ad errori,

3. Per rendere il programma più comprensibile a coloro che lo leggeranno e useranno, è necessario fornirlo di una consistente documentazione (vedi REMARK: commento).

A questi problemi si può porre riparo con l'uso dei sottoprogrammi. Un "sottoprogramma" è un programma completo indipendente, che può essere usato (cioè chiamato) dal programma principale o da altri sottoprogrammi.

7.1 **RETURN**

Non è necessario che una subroutine (sottoprogramma) cominci con un particolare enunciato: si può partire con un LET o con un FOR-TO, un INPUT ecc. L'ultimo, però, deve essere un enunciato RETURN, costituito semplicemente da un numero di linea e dal comando RETURN: con esso il controllo viene riportato al programma chiamante (in particolare alla linea che segue il punto di riferimento).

Sia ben chiaro che il controllo non può venire trasferito, dal sottoprogramma al programma chiamante, da altri tipi di enunciati condizionati (come l'IF-THEN o l'ON-GOTO).

Vediamo ora un esempio di sottoprogramma:

```
300 C1=(A+B+C)/3
310 C2=SQR(A∧2+B∧2+C∧2)
```

```
320 C3=SQR(A+B+C)
330 RETURN
```

Naturalmente prima che il sottoprogramma venga chiamato è necessario che le variabili A,B,C, siano già state definite.

Se desideriamo, all'interno di una subroutine vi possono essere più comandi RETURN; ecco ad esempio la struttura essenziale di un sottoprogramma di questo tipo:

```
100 ON N GOTO 200,300,400

200 X=A∧2+B
210 Y=INT(X)
220 RETURN

300 X=A∧3+C
310 RETURN

400 X=A∧3+C
410 RETURN

500 RETURN
```

Questo tipo di subroutine viene spesso chiamata ad estensione multipla. Appena s'incontra uno degli enunciati RETURN, il controllo viene riportato a quello che segue il riferimento alla subroutine.

7.2 **GOSUB**

Ad una subroutine ci si riferisce tramite l'enunciato GOSUB costituito da un numero di linea, dal comando GOSUB, e dal numero del primo enunciato del sottoprogramma suddetto.

Vediamo ora un esempio:

```
120 GOSUB 300
130 PRINT "Z=";Z


300 A=B  2
310 Z=EXP(A)
 . . . . . . . . . . . . .

400 RETURN
```

Quando nell'esecuzione di questo programma il calcolatore incontra l'enunciato 120 (GOSUB 300) il controllo viene trasferito al numero 300 e viene quindi eseguita la subroutine. Una volta raggiunta la linea 400 (RETURN) il controllo viene riportato al numero 130: il primo enunciato dopo GOSUB 300.

In generale, in un programma, ci possono essere più chiamate per lo stesso sottoprogramma; il controllo, eseguita la subroutine, torna sempre all'ultimo GOSUB che ha chiamato il suddetto sottoprogramma.

7.3 **ON-GOSUB**

Questa istruzione è molto simile all'istruzione ON-GOTO già vista in precedenza ed usa la stessa sintassi. La differenza sta nel fatto che con l'istruzione ON-GOSUB, in dipedenza del valore assunto dall'espressione numerica (opportunamente arrotondato), il controllo è trasferito alla corrispondente prima istruzione di un sottoprogramma e non in un qualsiasi punto del programma. Il calcolatore, poi, "ricorda" il numero di linea successivo alla ON-GOSUB, e vi ritorna alla fine del sottoprogramma stesso.

Naturalmente tutti i possibili sottoprogrammi ai quali si può arrivare con questa istruzione, devono terminare con il comando RETURN. Se così non fosse il calcolatore procederebbe in successione sino alla fine delle istruzioni che seguono il sottoprogramma e quindi l'istruzione ON-GOSUB si trasformerebbe in un semplice ON-GOTO.

7.4 **Funzioni definite dall'utente: l'enunciato DEF**

Abbiamo voluto inserire l'enunciato DEF in questo capitolo poiché, come si vedrà in seguito, esso, dal punto di vista logico, ha tutto l'aspetto di un sottoprogramma (pur essendo formato da una sola linea).

Abbiamo già visto che esistono in TI BASIC delle funzioni di libreria predefinite che snelliscono il lavoro di programmazione. Naturalmente, a seconda delle esigenze, può essere necessario ripetere certe particolari procedure (non previste nelle funzioni di libreria) durante la stesura di un programma. Per non specificare queste procedure ogni volta che servono, l'utente può definire una funzione tramite l'enunciato DEF e poi richiamarla in qualsiasi parte del programma.

Questo enunciato è così strutturato: numero di linea, comando DEF, nome della funzione, (parametro), =, espressione da definire. Per esempio:

```
100 DEF F(X)=2*X^2-3*X+4
 . . . . . . . . . . . . . .
200 C=F(3)-1
```

Innanzitutto c'è da dire che si possono definire in questo modo sia funzioni numeriche che di stringa; una funzione numerica restituisce, una volta calcolata, un valore numerico, un'alfanumerica una stringa. Questo enunciato, inoltre, serve solo per definire la funzione; essa viene calcolata solo nel momento in cui viene richiamata in qualche punto del programma usando opportunamente il nome che gli è stato assegnato (vedi esempio precedente: linea 200).

Il nome della funzione può essere un qualsiasi nome di variabile, ma se la funzione è di stringa si deve adottare il nome di unavariabile di stringa (cioè l'ultimo carattere deve essere $). Vediamo a questo punto due esempi.

Esempio 1.

```
100 DEF PC$(NOME$)=SEG$(NOME$,1,1)
110 INPUT "QUAL'È IL TUO NOME?":NOME$
120 PRINT "IL TUO NOME INIZIA CON LA LETTERA";PC$
130 GOTO 110
```

Esempio 2.

```
100 DEF A(X)=SQR(X^2-4)
110 S=A(3)
120 R=A(2)
130 PRINT S:R
140 END
```

Il parametro dell'enunciato DEF è un parametro fittizio come è evidente dagli esempi precedenti. Esso, infatti, ha carattere locale nel senso che il suo valore viene specificato nella linea in cui la funzione definita dalla DEF viene richiamata (vedi esempio 2: linee 110 e 120).

L'enunciato DEF può essere usato anche senza parametro, ma in questo caso la definizione si riduce ad una semplice assegnazione e, pertanto, ne sconsigliamo l'uso.

L'enunciato DEF può apparire ovunque in un programma ma deve sempre avere un numero di linea più basso del primo riferimento che s'incontra. Perciò, ed anche per chiarezza di lettura, si usa in genere raggruppare all'inizio del programma tutte le definizioni di funzioni che in esso compaiono.

ESERCIZI RISOLTI (CAP.7)

1) Può l'istruzione DEF avere un numero di linea qualunque?

Soluzione: No! L'enunciato DEF deve avere un numero di linea più basso dell'enunciato in cui viene richiamata per la prima volta.

2) Definire, in un programma TI BASIC, la funzione $y = ax^2 + bx + c$ dove a,b e c sono costanti che vengono assegnate all'inizio del programma ed x è la variabile indipendente.

Soluzione:

```
100 INPUT "DAMMI A,B,C": A,B,C
110 DEF Y (X)= A*X^2 + B*X + C
```

3) Scrivere un programma che trasferisca il controllo alla subroutine 1 se K è positivo ed alla subroutine 2 se K è negativo. Se K = 0 il programma si deve fermare.

Soluzione:

```
100 INPUT "DAMMI K":K
110 IF K  0 THEN 140
120 IF K  0 THEN 160
130 STOP
140 GOSUB 500
150 GOTO 130
160 GOSUB 1000
170 GOTO 130
```

4) I sottoprogrammi definiti dall'utente devono cominciare con un enunciato particolare?

Soluzione: No! Possono iniziare con un qualunque enunciato.

5) Identificare il tipo di errore nel seguente stralcio di programma:

```
100 GOSUB 1000
```

```
120 GOTO 1000
. . . . . . . . . . . .
1000 LET A=5
. . . . . . . . . . . .
1100 RETURN
```

Soluzione: L'errore è nella linea 120. Infatti non è possibile trasferire il controllo ad una subroutine tramite un enunciato GOTO.

ESERCIZI PROPOSTI (CAP. 7)

1) Spiegare in dettaglio la differenza tra l'enunciato DEF ed un sottoprogramma GOSUB-RETURN.

2) Che cos'è l'argomento locale di una DEF?
b) Che corrispondenza si deve avere tra argomento reale e argomento locale?

3) Spiegare le differenze tra gli enunciati ON-GOTO e ON-GOSUB e fare qualche programma che applichi il comando ON-GOSUB.

4) È possibile trasferire il controllo all'interno di una subroutine con un enunciato GO TO?

b) È possibile trasferire il controllo del programma da una subroutine all'esterno con un comando GOTO?

c) E con un IF-THEN?

5) Sfruttando l'enunciato ON-GOSUB, scrivere un programma che in dipendenza del valore assunto da un'opportuna variabile numerica, trasferisca il controllo a 2 o più subroutine.

CAPITOLO 8

Vettori e Matrici (Liste o Tabelle)

8.1 Concetto e costruzione di un Array in TI BASIC

Sia nei programmi gestionali che nei programmi scientifici è utile poter ordinare un insieme di dati in liste o tabelle e potersi riferire ad un singolo elemento di questa lista o tabella, come se fosse una variabile ordinaria.

Tali liste e tabelle (Array mono o bi-dimensionali) prendono anche il nome di vettori e matrici.

È possibile, in TI BASIC, trattare anche array tridimensionali.

Gli elementi degli array possono essere sia numerici che alfanumerici, ma nell'ambito dello stesso array è necessario ch'essi siano tutti dello stesso tipo.

Un esempio di vettore (lista) o array monodimensionale è il seguente:

```
M$(1)="MARE IN TEMPESTA"
M$(2)="MARE CALMO"
M$(3)="MARE MOSSO"
M$(4)="CONDIZIONI STAZIONARIE"
M$(5)="TEMPERATURA IN AUMENTO"
```

Tale lista, come si può osservare, è di tipo alfanumerico. Vediamo invece una tabella (o matrice) di tipo numerico:

```
B(1,1)= 1       B(1,2)= 0       B(1,3)=-5
B(2,1)= 7       B(2,2)=-1       B(2,3)= 0
B(3.1)=-3       B(3,2)= 0       B(3,3)=-7
```

I singoli elementi di un array si chiamano anche variabili indiciate (cioè con indice) o sottoscritte e si indicano con il nome dell'array seguito dal valore del'indice o degli indici (se si tratta di tabelle) tra parentesi.

Nel caso di una matrice gli indici indicano rispettivamente la riga e la colonna in cui l'elemento è collocato nella tabella.

Non è necessario che un indice sia una costante; si possono usare anche variabili o formule o nomi di funzione. È però importante ch'esso assuma

sempre un valore non negativo. Se, in quest'ultimo caso, il valore dell'espressione che costituisce l'indice non è intero, allora viene arrotondato troncando la parte decimale.

Vediamo ora come possiamo caricare in memoria il vettore M$ e la matrice B.

```
100 FOR I=1 TO 5
110 READ M$(I)
120 NEXT I
130 FOR I=1 TO 3
140 FOR J=1 TO 3
150 READ B(I,J)
160 NEXT J
170 NEXT I
180 DATA "MARE IN TEMPESTA", "MARE CALMO", "MARE MOSSO"
190 DATA "CONDIZIONI STAZIONARIE","TEMPERATURA IN AUMENTO"
    1,0,-5
200 DATA 7,-1,0,-3,0,-7
```

Certamente un notevole vantaggio, usando un array, è quello di poter scrivere una sola assegnazione all'interno di un ciclo FOR-TO-NEXT. Naturalmente è possibile fare la stessa cosa usando molteplici assegnazioni (ad esempio 14 nel precedente esempio) ma la cosa risulta alquanto lunga e noiosa.

8.2 **DIM**

Nel TI BASIC è possibile lavorare con array i cui indici variano tra 0 e 10; quindi si possono usare liste di 11 elementi, tabelle di 121 elementi (11 × 11) e array tridimensionali di 1331 elementi (11 × 11 × 11). Ciò non vuol dire che non si possa lavorare con array i cui indici variano da 0 a oltre 10! Ma per far ciò dobbiamo avvertire il calcolatore con l'enunciato dichiarativo DIM (DIMENSION: dimensione).

La sintassi di questa istruzione è la seguente: numero di linea, enunciato DIM, lista degli array separati da virgole e con una o più costanti tra parentesi che indicano la massima variabilità degli indici.

Supponiamo ad esempio di voler usare un vettore a 70 componenti ed una tabella di 22 righe e 20 colonne; la nostra dichiarazione sarà allora la seguente:

```
100 DIM V(70), M(22,20)
```

dove V è il nome dell'array monodimensionale ed M di quello bidimensionale.

Questo tipo di dichiarazione deve essere sempre fatto prima di qualunque riferimento agli array contenuti nel programma; si preferisce, quindi, porle sempre all'inizio del programma stesso.

8.3 **OPTION BASE**

Nel TI BASIC esiste un istruzione che permette di definire uguale ad 1 il valore inferiore dell'indice che altrimenti è considerato 0.

L'istruzione OPTION BASE 1, che serve allo scopo, deve essere posta sempre prima dell'enunciato DIM e vale per tutti gli array presenti nel programma.

Concludiamo questo capitolo illustrando l'uso dell'istruzione RESTORE nel caso di rilettura dati di array. In particolare facciamo rileggere un blocco DATA per assegnare i valori ad un altro array.

```
100 DIM A (20), B (15)
110 FOR I=1 TO 20
120 READ A (I)
130 NEXT I
140 RESTORE
150 FOR I=1 TO 15
160 READ B (I)
170 NEXT I
180 DATA 1,2,-7,-0.5,-13
190 DATA 3.15,-3,-18.3,15,-4
200 DATA 32,-7,0.68,1.1,-2
210 DATA 3,-73,12,1.127,-7
```

Il primo ciclo (linee 110, 120, 130) assegna all'array A i 20 dati contenuti nel blocco DATA (da 180 a 210). Le istruzioni che vanno dalla linea 140 alla linea 170, assegnano al vettore B i valori contenuti nel blocco DATA dalla linea 180 alla linea 200. Se non ci fosse stata l'istruzione 140 (RESTORE), il calcolatore avrebbe segnalato un errore del tipo "DATA ERROR" poiché avendo ormai assegnato tutti i valori contenuti nei DATA al primo vettore A, non ci sarebbero più stati valori da assegnare al vettore B; il comando RESTORE ha permesso, invece, la rilettura dei dati a partire dal primo valore del primo DATA (linea 180).

ESERCIZI RISOLTI (CAP.8)

1) Caricare in un vettore a 20 componenti i valori numerici contenuti in un blocco DATA.

Soluzione:

```
100 DIM V(20)
110 FOR I=1 TO 20
120 READ V(I)
130 NEXT I
140 DATA 3,5,21,1,0,4,7,-1,1,0
150 DATA 11,11,14,16,0,0,-1,3,-1,9
```

Ricordiamo che l'enunciato DIM è necessario tutte le volte in cui si vuole dimensionare un «array» monodimensionale o vettore a più di 11 componenti.

2) Facciamo ora la stessa operazione per un «array» alfanumerico.

Soluzione:

```
100 DIM V$(20)
110 FOR I=1 TO 20
120 READ V$(I)
130 NEXT I
140 DATA MARIO,PAOLO,FRANCO,ANTONIO,MAURO,PINO
150 DATA FAUSTO,VINCENZO,MARCO,UGO,GABRIELE,ALFREDO,GINO
160 DATA VITO,LINO,CARLO,ANDREA,LUIGI,STEFANO,DINO
```

3) Se volessimo stampare il vettore V$ del programma precedente, quali linee dovremmo aggiungere?

Soluzione:

```
170 FOR I=1 TO 20
180 PRINT V$(I),
190 NEXT I
```

4) Supponiamo di avere una tabella numerica del seguente tipo:

7	5	8	1	3	4	6
1	2	1	4	0	1	0
9	3	4	8	8	7	8

carichiamola in un array bidimensionale e sommiamo gli elementi di ciascuna riga caricandoli rispettivamente in tre variabili numeriche S1, S2 e S3.

Soluzione:

```
100 DIM A (3,7)
110 FOR I=1 TO 3
120 FOR J=1 TO 7
130 READ A(I,J)
140 NEXT J
150 NEXT I
160 DATA 7,5,8,1,3,4,6
170 DATA 1,2,1,4,0,1,0
180 DATA 9,3,4,8,8,7,8
190 I=1
200 FOR J=1 TO 7
210 S1=S1+A(I,J)
220 NEXT J
230 I=2
240 FOR J=1 TO 7
250 S2=S2+A(I,J)
260 NEXT J
270 I=3
280 FOR J=1 TO 7
290 S3=S3+A(I,J)
300 NEXT J
310 PRINT S1,S2,S3
```

5) Preparare una stampa che visualizzi il cognome ed il nome di 5 persone con la relativa età. È necessario che tra una linea di stampa ed un'altra vi sia una linea bianca. I dati sono: MARIO ROSSI 30, GINO FEDE 41, MAURO BIANCHI 19, FRANCO LA TEANA 81, ADOLFO NOTO 72;

Soluzione:

```
100 DIM A$(5),E(5)
110.FOR I=1 TO 5
```

```
120 READ A$(I),E(I)
130 NEXT I
140 DATA "ROSSI MARIO",30,"FEDE GINO",41
150 DATA "BIANCHI MAURO",19,"LA TEANA FRANCO",81
160 DATA "NOTO ADOLFO",72
170 PRINT "COGNOME E NOME";TAB(25);"ETÀ"
180 FOR I=1 TO 5
190 PRINT:A$(I);TAB(25);E(I)
200 NEXT I
```

ESERCIZI PROPOSTI (CAP.8)

1) Spiegare la funzione dell'enunciato OPTION BASE;
b) È necessario l'uso della DIM nel precedente esercizio n°5?

2) È necessario dimensionare una matrice di 7 righe e 12 colonne?

3) Generare un vettore a 100 componenti in cui in ogni componente vi sia il quadrato dell'indice corrispondente.

4) Quali limitazioni vi sono ai valori che un indice di array può assumere?

5) Mediante gli enunciati READ e DATA, assegnare al vettore L$ e alla matrice T i seguenti valori:
L$(1)=BIANCO; L$(2)=GIALLO; L$(3)=ROSSO; L$(4)=VERDE;
T(1,1)=-1; T(1,2)=-3; T(1,3)=6; T(1,4)=-7; T(2,1)=-2; T(2,2)=5; T(2,3)=-8; T(2,4)=8;
e farli stampare.

CAPITOLO 9

Introduzione alle subroutine speciali del TI 99/4A

Oltre al BASIC, fin qui trattato, le macchine della quarta generazione dispongono di funzioni speciali particolari che le rendono diverse tra di loro.
Queste funzioni (che in genere sono dei veri e propri sottoprogrammi) riguardano la grafica, il colore ed il suono. È naturale, quindi, aspettarsi che funzioni di questo tipo siano diverse da macchina a macchina. A parte la necessità di dover scrivere su monitor di diverse dimensioni, queste funzioni sono anche diverse perché legate al particolare tipo di sistema operativo che la macchina usa.
È evidente, quindi, che ciò che verrà detto nelle pagine che seguono, è valido solo per il TI 99/4A.
Parliamo ora della grafica.

9.1 **CALL CLEAR**

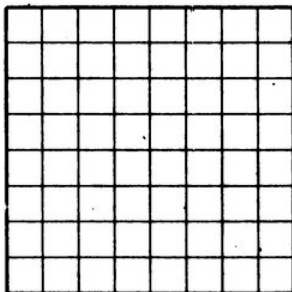
Quando normalmente si lavora con la grafica è importante, prima di iniziare l'esecuzione di un programma, poter cancellare dallo schermo tutti i caratteri precedentemente scritti. Il sottoprogramma che si occupa di ciò è il CALL CLEAR.
Questo comando è sempre la prima istruzione di qualsiasi programma di grafica, e può essere usato comunque in ogni tipo di programma.
Proviamo infatti ad eseguire il seguente programma:

```
100 CALL CLEAR
110 PRINT "GRAFICA A COLORI"
```

Osserviamo infatti, che prima di stampare il messaggio contenuto nell'istruzione PRINT, il calcolatore cancella tutto ciò che risiede sullo schermo compreso il listato del programma appena battuto.
In seguito saremo in grado di apprezzare l'uso di questa subroutine fino a trovarla addirittura indispensabile.

9.2 CALL CHAR

Ci occupiamo della grafica vera e propria del nostro computer.

Il cursore, che normalmente vediamo lampeggiare sul nostro monitor, è in realtà una matrice di 8×8 punti e per comodità la riportiamo qui sotto ingrandita.

Ciascuno dei 64 puntini che compongono la suddetta matrice, può accendersi o spegnersi indipendentemente dagli altri. Ma come si può ottenere ciò? Supponiamo di dividere la nostra matrice in 8 righe ognuna, ovviamente, composta da 8 piccoli punti. Dividiamo poi ogni riga in due parti uguali (cioè di 4 punti ciascuna) e numeriamo queste mezze righe così ottenute. L'ordine di numerazione che adottiamo è il seguente: 1 per la prima mezza riga, 2 per la seconda mezza riga in alto a destra, 3 per la mezza riga a sinistra subito sotto la numero 1, e così via dicendo fino alla numero 16 rappresentata dall'ultima mezza riga in basso a destra.

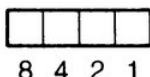
Ora, ad ognuna di queste mezze righe così ottenute, dobbiamo assegnare un carattere esadecimale a seconda del numero e di quali punti vogliamo tenere accesi. Voglio brevemente citare la corrispondenza che c'è tra i numeri decimali e quelli esadecimali.

DECIMALI :	0	1	2	3	4	5	6	7	8	9	10	11	12	13
ESADECIM.:	0	1	2	3	4	5	6	7	8	9	A	B	C	D

DECIMALI :	14	15
ESADECIM.:	E	F

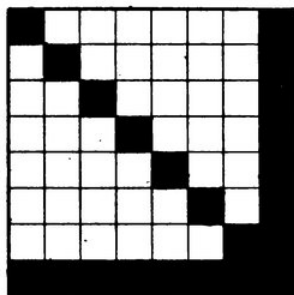
Osserviamo che i primi 10 simboli dell'alfabeto esadecimale sono uguali ai ben noti simboli dell'alfabeto decimale. Vengono poi introdotti i simboli A B C D F E per rappresentare i numeri da 10 a 15.

Torniamo ora all'argomento che più ci interessa. Prendiamo una delle 16 mezze righe ed assegniamo a ciascuno dei 4 puntini che la compongono un

peso numerico come indicato in figura.

Dopodichè dovremo moltiplicare per 1 il peso delle caselle che vogliamo accendere e per 0 quello delle caselle che vogliamo spegnere (o tenere spente). Fatto ciò dovremo sommare i numeri così ottenuti e tradurre il tutto in esadecimale. Un esempio, comunque, aiuterà meglio a capire la cosa: supponiamo di voler accendere il 2° ed il 4° punto, come indicato in figura.

Il numero che otteniamo è:
$A = 1 \times 8 + 0 \times 4 + 1 \times 2 + 0 \times 1 = 10$
Oppure supponiamo di voler accendere il 2°, il 3° e il 4° punto come mostrato in figura.

Il numero che allora si ottiene è:
$A = 1 \times 8 + 1 \times 4 + 1 \times 2 + 0 \times 1 = 14$ che in esadecimale è E.

A questo punto abbiamo capito come dobbiamo fare per assegnare un carattere (o meglio una stringa) esadecimale a ciascuna mezza riga. Vediamo quindi come formare un carattere completo. Supponiamo che il carattere da rappresentare sia quello in figura:

e facile verificare che la sequenza di caratteri che lo rappresenta è la seguente:

81412111090503FF

N.B.: È molto importante l'ordine con cui si succedono i precedenti caratteri.

Trovata la rappresentazione esadecimale del carattere che vogliamo far apparire sullo schermo, dobbiamo trasmetterlo al calcolatore. Per fare questo dobbiamo usare la seguente istruzione:

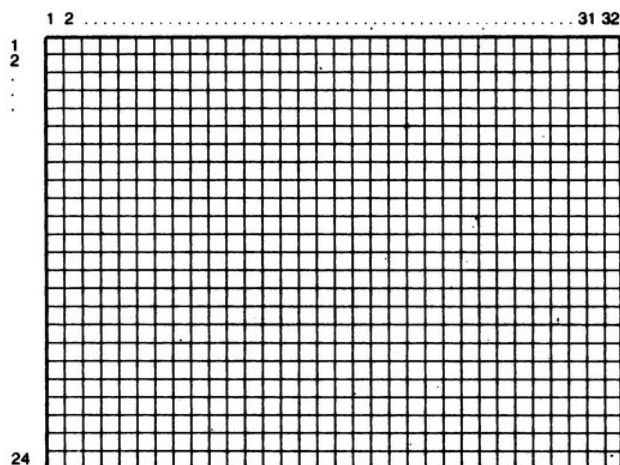
CALL CHAR (N,"81412111090503FF")

dove N è un numero compreso tra 128 e 159 che prende il nome di codice del carattere ed è il «nome» per poterlo distinguere da altri; la stringa che segue (si tratta infatti di una stringa perché è alfanumerica) prende il nome di identificatore di sagoma.

Dobbiamo fare attenzione che questo sottoprogramma serve solo per definire un carattere; per poterlo visualizzare sul monitor dobbiamo immettere nel programma anche un'istruzione PRINT. Cioè:

```
100 CALL CHAR (130,"81412111090503FF")
110 PRINT CHR$(130)
```

9.3 **CALL HCHAR**

Supponiamo ora di voler disegnare in un punto ben preciso dello schermo il carattere che abbiamo generato e catalogato nella CALL CHAR. È ovvio che a questo punto dobbiamo sapere qualcosa di più sul nostro schermo. È anch'esso una matrice (questa volta non quadrata) di 32 colonne e 24 righe disposte nel seguente modo:

Dobbiamo però considerare solo 28 colonne in quanto le prime 2 e le ultime 2 non sono visibili sul video del televisore (ma del monitor si!).
Vediamo ora come è fatta la struttura di un enunciato CALL HCHAR:

CALL HCHAR (N,M,P,D)

dove N indica il numero di riga, M il numero di colonna e P il codice carattere con cui abbiamo chiamato il nostro carattere nella CALL CHAR. L'ultimo argomento, D, è opzionale ed indica il numero di ripetizioni in orizzontale.
Proviamo ad eseguire il seguente programma:

```
100 CALL CLEAR
110 CALL HCHAR (1,3,36,28)
```

Osserviamo che la prima riga si riempie di $. Infatti 36 è il codice ASCI di questo carattere.

9.4 **CALL VCHAR**

La struttura di una CALL VCHAR è identica a quella di una CALL HCHAR. La sola differenza sta nell'argomento D: cioè il numero di ripetizioni è fatto verticalmente invece che orizzontalmente. Ci possiamo rendere conto di ciò scrivendo il seguente programma:

```
100 CALL CLEAR
110 CALL VCHAR(3,5,37,15)
```

9.5 **CALL GCHAR**

Si può dire che questa funzione compie l'operazione inversa delle CALL HCHAR e della CALL VCHAR.
Infatti quando vogliamo leggere il codice ASCII di un carattere disegnato sul video, dobbiamo ricorrere al sottoprogramma CALL GCHAR. Il suo enunciato generale è:

CALL GCHAR (N,M,X)

dove N e M indicano le posizioni cartesiane del carattere che vogliamo leggere sul video ed X è il nome di una variabile numerica nella quale il calcolatore scrive il codice ASCII del suddetto carattere. Vediamo un programma che ne fa uso:

```
100 CALL CLEAR
110 FOR K=1 TO 10
120 CALL HCHAR (K,5,65,7)
130 CALL GCHAR (3,8,X)
140 PRINT "X=";X
```

Questa sequenza di istruzioni genera una serie di righe (10) e di colonne (7) di caratteri A. Poi viene letto il codice ASCII del carattere che sullo schermo occupa la posizione 3,8 e viene quindi copiato nella variabile X. Tale valore è poi stampato mediante una PRINT.

9.6 **CALL SCREEN**

Con l'uso di questo sottoprogramma possiamo cambiare il colore dello schermo, durante l'esecuzione di un programma, a nostro piacimento. I codici per il colore sono qui riportati.

CODICE COLORE	COLORE
1	trasparente
2	nero
3	verde
4	verde chiaro
5	bleu scuro
6	bleu chiaro
7	rosso scuro
8	viola
9	rosso
10	rosso scuro
11	giallo scuro
12	giallo chiaro
13	verde scuro
14	magenta
15	grigio
16	bianco

L'espressione generale di questo sottoprogramma è:

CALL SCREEN (numero o espressione numerica)

L'espressione numerica viene calcolata ed eventualmente arrotondata ad intero; comunque il numero o il risultato dell'arrotondamento dell'espressione numerica deve essere compreso tra 1 e 16.

Durante la normale esecuzione di un programma, quando non è enunciata questa subroutine, il colore dello schermo è verde chiaro (codice colore 4). Il programma che segue mostra le possibili variazioni di colore dello schermo:

```
100 CALL CLEAR
110 FOR I=1 TO 16
120 CALL SCREEN (I)
130 FOR K=1 TO 100
140 NEXT K
150 NEXT I
```

9.7 **CALL COLOR**

Questo programma permette di dare un diverso colore a più caratteri, grafici e non, facenti parte di un disegno. La espressione generale è:

CALL COLOR (numero dell'insieme dei caratteri, codice per il colore del carattere, codice colore per lo sfondo del carattere)

I parametri tra parentesi sono tutti numeri o espressioni numeriche. Per ogni carattere visualizzato si possono specificare due colori: il colore dei punti che lo formano (codice colore carattere) e quello che occupa la parte restante non accesa (codice colore sfondo carattere). Per ragioni tecniche, che qui non stiamo a spiegare, i 159 caratteri ASCII sono stati suddivisi in 16 gruppi ad ognuno dei quali è stato assegnato un numero chiamato numero d'insieme.

Quando usiamo la CALL COLOR dobbiamo specificare a quale dei 16 insiemi di caratteri appartiene quello che stiamo stampando. Quella che segue è la tabella degli insiemi dei caratteri:

CODICI CARATTERE	NUMERO D'INSIEME
32— 39	1
40— 47	2
48— 55	3
56— 63	4
64— 71	5
72— 79	6
80— 87	7
88— 95	8
96—103	9
104—111	10
112—119	11
120—127	12
128—135	13
136—143	14
144—151	15
152—159	16

L'esempio che segue mostra l'uso della CALL COLOR.

```
100 CALL CLEAR
110 CALL COLOR (5,7,4)
120 CALL COLOR (6,3,4)
130 CALL COLOR (7,5,4)
140 PRINT "AIUOLA"
150 GOTO 150
```

Ogni carattere avrà il colore specificato dalla rispettiva CALL COLOR.

9.8 **CALL SOUND**

Le capacità sonore del TI 99/4A sono, a nostro avviso, veramente eccellenti. Ormai quasi tutti i microcalcolatori, del genere TI 99/4A, possono suonare; ma non tutti possono suonare più d'una nota contemporaneamente ossia un accordo. La forma dell'enunciato, in generale, che chiama questo sottoprogramma è:

CALL SOUND (durata, frequenza 1, volume 1, frequenza 2, volume 2, frequenza 3, volume 3)

Nel caso in cui volessimo suonare una sola nota, si ometteranno le frequenze 1 e 2 ed i rispettivi volumi. Nella tabella riportata sul manuale del TI 99/4A sono riportate tutte le corrispondenze tra frequenze e note che il TI può suonare. Il volume è un numero che varia da 0 a 30, mentre la durata viene specificata in millisecondi e può variare tra 1 e 4250 estremi compresi. Se per la frequenza viene dato un valore negativo, viene prodotto un rumore anziché un suono. I rumori possibili sono 8 e per ascoltarli si può provare ad eseguire questo programma:

```
100 FOR I=-8 TO -1
110 CALL SOUND (1500,I,5)
120 NEXT I
```

Questi rumori prodotti dal calcolatore, vengono essenzialmente usati come effetti sonori nei «video games». Per riuscire ad usarli efficientemente, è necessario che l'utente si eserciti molto.

9.9 **CALL KEY**

Abbiamo visto che con il comando INPUT si possono introdurre dati (numeri, stringhe, ecc.) in un programma tramite tastiera. Ma questa istruzione, da un certo punto di vista ha degli inconvenienti. Ogni volta che il calcolatore incontra un comando INPUT stampa sul video un punto interrogativo, inoltre stampa sul video anche il dato (o i dati) che noi introduciamo e ad ogni stampa (sia di punto interrogativo che di dato) avviene lo scrolling di una riga In alcuni programmi questo può essere un vero inconveniente: immaginiamo per esempio un gioco dove la grafica occupa la maggior parte del video!

A ciò, in parte, può rimediare il sottoprogramma KEY poiché permette d'introdurre un dato da tastiera senza disturbare le informazioni presenti sul video. Diciamo in parte poiché mentre con l'istruzione INPUT si può introdurre da tastiera stringhe di caratteri, con il sottoprogramma KEY si può introdurre solo un carattere alla volta.

Vediamo ora come adoperare questa subroutine.

CALL KEY (unità di tastiera, variabile di ritorno, variabile di stato).

L'unità di tastiera può essere un numero o un espressione numerica che deve avere un valore compreso tra 0 e 5. Essa serve per specificare in che modo intendiamo usare la tastiera e cioè:

usando l'unità 0, la macchina accetta tutti i caratteri della tastiera e scrive il codice ASCII del carattere battuto nella variabile di ritorno;
se si usa l'unità 1, la macchina accetta dall'esterno solo i caratteri che sono situati nella parte sinistra della tastiera (la linea di demarcazione tra la parte destra e la parte sinistra della tastiera sta tra i caratteri: 5-6, T-Y, G-H, B-N;
se si usa l'unità 2, invece, essa accetta dall'esterno solo i caratteri che sono situati nella parte destra della tastiera:
le unità 3,4 e 5, ridefiniscono la tastiera per operazioni particolari usando codici diversi da quello ASCII. Data la particolarità di queste unità, non riteniamo opportuno trattarle in questa sede.

La variabile di ritorno deve essere una variabile numerica, in essa infatti il calcolatore andrà a porre il codice ASCII del carattere battuto, se si è scelta l'unità di tastiera 0, oppure opportuni numeri da 0 a 19, se si sono scelte le unità di tastiera 1 o 2. Ad esempio se scegliamo l'unità 0 e poi battiamo il carattere A sulla tastiera, sullo schermo non compare nulla ma nella variabile di ritorno il calcolatore scrive 65 (che come sappiamo è il codice ASCII del carattere A).

La variabile di stato è anch'essa una variabile numerica e serve da controllo; infatti indica al calcolatore ciò che è avvenuto sulla tastiera ed in particolare, dopo l'esecuzione della CALL KEY, la macchina pone in questa variabile il valore:

- 0 quando non è stato premuto alcun tasto;
- +1 appena viene premuto un tasto per la prima volta, oppure quando successivamente si premerà un tasto diverso dal precedente;
- -1 quando il tasto battuto è uguale al precedente.

Facciamo ora un esempio. Supponiamo di voler fare un programma che gestisce l'archivio di una libreria. Una possibile strutturazione logica del programma è la seguente. La prima pagina video elenca i settori in cui è divisa la libreria affiancati ognuno dal numero che bisogna battere per visualizzare i libri contenuti in quel settore. Una volta che questa pagina appare sul monitor, il calcolatore attende che l'utente scelga uno di questi settori. È a questo punto che interviene l'uso della CALL KEY! Infatti, quando viene premuto il tasto di scelta, può essere controllato ciò che accade sulla tastiera tramite la variabile di stato e quindi, usando tanti IF-THEN quanti sono i settori, indirizzare il programma nel settore desiderato dall'utente.

Vediamo qual'è la struttura essenziale che risolve tale problema quando il numero dei settori è 5.

```
100 CALL CLEAR                }
110 ........................  } presentazione video dei 5 settori
   .........................  }
```

```
200 CALL KEY (0,K,S)
210 IF S=0 THEN 200
220 IF K=49 THEN 500
230 IF K=50 THEN 1000
240 IF K=51 THEN 1500
250 IF K=52 THEN 2000
260 IF K=53 THEN 2500
270 IF K>53 THEN 200
280 IF K<49 THEN 200
. . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

(linee 220–260) le 5 istruzioni che indirizzano verso i settori

(linee 270–280) controllo per non accettare tasti diversi da 1,2,3,4, e 5

```
500 CALL CLEAR
510 . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

presentazione video del primo settore

```
1000 CALL CLEAR
1010 . . . . . . . . . . . . . . . . . . . . . . . .
```

presentazione video del secondo settore

La struttura «diramante» che va dalla linea 200 alla linea 280 può essere anche sostituita con la seguente:

```
200 CALL KEY (0,K,S)
210 IF S=0 THEN 200
220 IF K<49 THEN 200
230 IF K>53 THEN 200
240 ON(54-K)GOTO 2500,2000,1500,1000,50
```

Spieghiamo il funzionamento della prima struttura. Fino a quando nessun tasto viene battuto, il calcolatore continua ad eseguire alternativamente le linee 200 e 210; infatti, essendo S=0, non può uscire dal loop. Quando battiamo un qualsiasi tasto, nella variabile K viene posto il codice ASCII relativo al tasto battuto. Siccome a noi interessa che l'utente batta un tasto da 1 a 5, dobbiamo fare in modo che se ciò non accade il programma trasferisca il controllo alla linea 200. Questo compito viene assolto dalle linee 270 e 280. Le linee che vengono prima di queste servono ad indirizzare verso una parte precisa del programma in dipendenza del tasto battuto.

Il funzionamento della seconda struttura si basa sulla istruzione ON-GOTO. Questa volta gli IF-THEN delle linee 220 e 230, che corrispondono alle linee 270 e 280 della prima struttura, sono posti prima dell'istruzione ON-GOTO per il motivo che ora spiegheremo. Se il tasto battuto ha un codice ASCII maggiore di 54, l'espressione (54-K) è ovviamente negativa. Se K è minore di 49, l'espressione suddetta produce un numero maggiore di 5. In entrambi i casi, allora, l'istruzione ON-GOTO non può essere eseguita ed il calcolatore si blocca stampando un messaggio di errore del tipo: "BAD VALUE in 240".

ESERCIZI RISOLTI (CAP. 9)

1) Scrivere un programma che mostri tutti i colori che lo schermo del TI 99/4A può assumere.

Soluzione:

```
100 CALL CLEAR
110 FOR I=1 TO 16
120 FOR T=1 TO 200
130 NEXT T
140 CALL SCREEN (I)
150 NEXT I
```

N.B.: lo scopo delle linee 120 e 130 è quello di far ritardare la sequenza dei colori mostrati; infatti, molto spesso, esso prende il nome di «ciclo di perdita».

2) Mediante le funzioni CALL CHAR, CALL HCHAR e CALL VCHAR, riprodurre sullo schermo una sottile cornice di 10×10 cursori.

Soluzione:

```
100 CALL CLEAR
110 CALL CHAR (130,"0101010101010101")
120 CALL CHAR (131,"8080808080808080")
130 CALL CHAR (132,"00000000000000FF")
140 CALL CHAR (133,"FF")
150 CALL HCHAR (7,7,132,10)
160 CALL VCHAR (8,6,130,10)
170 CALL HCHAR (18,7,133,10)
180 CALL VCHAR (8,17,131,10)
```

N.B.: l'identificatore di sagoma della linea 140 (FF), è una abbreviazione di "FF00000000000000"; infatti quando la macchina riceve una stringa i cui caratteri sono inferiori a 16, assegna di default (si potrebbe dire d'ufficio), per i restanti caratteri, il carattere 0.

3) Illustrare come sia possibile bloccare il programma ad un enunciato CALL KEY, finché non viene battuto un tasto

Soluzione: Ciò è evidente se si pensa ai valori che può assumere la variabile di STATUS (che è la terza tra parentesi; infatti se STATUS=0 vuol dire che nessun tasto è stato battuto. È ovvio, allora, che dovremo scrivere:

```
100 CALL KEY (0,K,S)
110 IF S=0 THEN 100
```

Infatti appena viene battuto un qualunque tasto sulla tastiera, il calcolatore scrive nella variabile S il valore +1 (o -1) ed il programma va avanti.

4) Mostrare quali linee dobbiamo aggiungere al programma dell'esercizio 2 per dare differenti colori a quella cornice.

Soluzione: Si tratta di una sola linea da inserire a 190.

```
190 CALL COLOR (13,10,4)
```

Infatti 13 è l'insieme dei caratteri che vanno da 128 a 135 (proprio dove sono compresi i codici di carattere 130, 131, 132 e 133). Inoltre, a causa del secondo e terzo argomento del comando CALL COLOR, si avrà una cornice rossa su sfondo verde.

5) Elaborare un programma che, chiedendo il nome di una nota nell'ambito di una scala, produca poi il suono corrispondente.

```
100 INPUT "DAMMI UNA NOTA":A$
110 IF A$="DO" THEN 180
120 IF A$="RE" THEN 200
130 IF A$="MI" THEN 220
140 IF A$="FA" THEN 240
150 IF A$="SOL" THEN 260
160 IF A$="LA" THEN 280
170 IF A$="SI" THEN 300
180 A=262
190 GOTO 310
```

```
200 A=294
210 GOTO 310
220 A=330
230 GOTO 310
240 A=349
250 GOTO 310
260 A=392
270 GOTO 310
280 A=440
290 GOTO 310
300 A=494
310 CALL SOUND (200,A,4)
320 GOTO 100
```

6) Generare un programma che sia in grado di produrre un suono compreso tra le frequenze 200 e 800 e con un volume regolabile tramite una variabile numerica compresa tra 0 e 30.

Soluzione:

```
100 INPUT "DAMMI UN NUMERO TRA 200 e 800":N
110 INPUT "DAMMI IL VOLUME (0,30)":V
120 CALL SOUND (1000,N,V)
130 GOTO 100
```

(Lo spirito di questo esercizio, come d'altronde di molti altri in precedenza, è quello di far capire all'utente come il linguaggio TI BASIC sia molto adatto per «conversare» con la macchina).

ESERCIZI PROPOSTI (CAP.9)

1) Qual'è la differenza tra l'istruzione CALL CLEAR (inserita come comando immediato) ed il comando NEW visto in precedenza?

2) Illustrare il funzionamento della CALL GCHAR e fare un programma che la usi.

3) Spiegare in dettaglio quali sono le differenze tra i sottoprogrammi CALL HCHAR e CALL VCHAR.

4) Far girare il seguente programma e spiegare:

a) Il significato dell'istruzione 110.

b) Il perché del loop 140...150.

c) La necessarietà o meno del comando END.

```
100 A$="8080B0C88484C8B0"
110 CALL CHAR (96,A$)
120 CALL CLEAR
130 CALL HCHAR (5,12,96,10)
140 FOR H=1 TO 100
150 NEXT H
160 GOTO 120
170 END
```

d) Che cosa accade se la linea 130 viene sostituita nel seguente modo:

```
130 CALL VCHAR (5,12,96,10)
```

5) Fare un programma che esegua gli accordi di DO maggiore (DO, MI, SOL), FA maggiore (FA,LA,DO). SOL maggiore (SOL,SI,RE). Usare la tabella delle

corrispondenze nota-frequenza riportata sul manuale del TI 99/4A.

6) Usando le istruzioni CALL KEY, ASC, ON-GOTO e altre, fare un programma che visualizzi le pagine della vostra agenda telefonica. Consiglio: a parte la presentazione grafica, che è lasciata a vostro libero arbitrio, costruire la struttura essenziale del programma nel modo indicato:

```
100 INPUT "QUALE LETTERA DESIDERI?":A$
110 K=ASC (A$)
120 ON (K-64) GOTO 500,600,700,.......
```

dove le linee 500,600, ecc. riferiscono il programma sulle varie pagine video della vostra agenda.
Se non si vuole che la lettera da voi inserita non venga riportata sul video, eseguire la trasformazione seguente:

```
100 PRINT "QUALE LETTERA DESIDERI?":::::
110 CALL KEY(O,K,S)
120 ON (K-64) GOTO 500,600,700.....
```

Buona Fortuna!!!

CAPITOLO 10

Introduzione alle tecniche di programmazione

Abbiamo già avuto modo di definire un programma come una serie di istruzioni che il calcolatore interpreta nel suo linguaggio (il linguaggio macchina) ed esegue una dopo l'altra.

Abbiamo anche mostrato come impostare le istruzioni per ottenere determinati risultati e quali sono i comandi da usare in TI BASIC.

All'inizio sembra abbastanza facile, con queste conoscenze, arrivare a dei risultati concreti. Ma, quando i problemi che si debbono risolvere non sono così semplici da permettere un'immediata trascrizione della loro soluzione in un linguaggio di programmazione, sorgono le prime difficoltà: bisogna riflettere a lungo per trovare una soluzione valida, fare numerosi tentativi ed una volta trovato il metodo di risoluzione cercare di ottimizzarlo.

È facile immaginare che per un dato problema non vi è un solo metodo di risoluzione. Dovendone, però, scegliere uno, è necessario adottare dei giusti criteri. Generalmente un programma si considera efficiente, cioè ottimizzato, se contiene il minor numero possibile di istruzioni, se porta velocemente al risultato voluto e se occupa il minimo spazio di memoria. Non sempre, però, è facile valutare tra due programmi qual'è il migliore: ovviamente quello che ha più istruzioni occupa più memoria, ma potrebbe risultare più veloce, in esecuzione, di quello più breve.

Ottimizzare un programma, inoltre, ha come conseguenza una minore chiarezza del listato ed una riduzione di documentazione, cosa da non sottovalutare poiché implica una notevole perdita di tempo da parte di colui che dovrà leggere ed usare il programma (questo concetto si capirà meglio studiando il comando REM).

A questo punto è chiaro che bisogna scegliere un ragionevole compromesso tra tutti i vantaggi e gli svantaggi. Facciamo un esempio. Se in un programma si deve elevare a potenza una variabile, ad esempio X∧2, si possono adottare due metodi:

1) scrivere X∧2
2) scrivere X*X

Il primo è senz'altro più chiaro (si individua subito il tipo di operazione!); di contro il secondo è più efficiente (fa risparmiare tempo macchina) ma, soprattutto per potenze maggiori, è poco chiaro. Allora si preferisce scrivere l elevazione nel modo 2) quando si eleva al quadrato, nel modo 1) in tutti gli altri casi.

Ci sono varie tecniche che guidano ad un corretto lavoro di programmazione come ad esempio la programmazione strutturata ed il metodo TOP-DOWN. In questo libro non intendiamo entrare nei dettagli di queste tecniche poiché l'argomento è molto vasto e specialistico, ma piuttosto parlare di alcuni algoritmi (cioè metodi di risoluzione) che risolvono problemi specifici, di diagrammi di flusso che aiutano a progettare un programma e di alcuni accorgimenti che il programmatore deve adoperare se vuole ottenere un programma efficiente e/o documentato.

10.1 **Diagrammi di flusso**

Una delle tecniche che il programmatore usa per impostare correttamente un programma è quella di fare, prima della stesura vera e propria, un'analisi grafica del problema mediante i diagrammi di flusso o «flow chart».

Lo scopo di questo paragrafo è quello d'insegnare all'utente come costruire una flow chart; premettiamo che essa va sempre costruita in modo indipendente dal tipo di linguaggio che si userà poi per tradurla.

Fare un diagramma di flusso consiste nello scrivere, in linguaggio corrente, all'interno di particolari simboli, uno dopo l'altro tutti i passaggi logici che il calcolatore deve fare per risolvere un problema. In particolare ogni operazione (o gruppo di operazioni) aritmetica o logica va racchiusa entro particolari figure geometriche (stabilite nella convenzione AFNOR Z 67-010 dell'aprile 1966), collegate tra loro mediante linee orientate che rappresentano il flusso delle informazioni all'interno del programma. Tipi diversi di operazioni si indicano con forme geometrice diverse. Vediamo i principali simboli ed il loro uso.

SIMBOLI	USO
	Serve ad evidenziare l'inizio e la fine di un programma.
	Per un calcolo o per un procedimento che però non sia una decisione.
	Indica una decisione: ad essa si collegano sempre 3 o più linee.
	Per operazioni d'ingresso e uscita.

Chiamata a sottoprogramma.

Serve per indicare la continuazione del programma in un altra pagina.

Collega logicamente due proposizioni: la freccia indica il verso in cui si propaga l'informazione.

Qualche esempio a questo punto può chiarire meglio ciò che è stato appena detto.

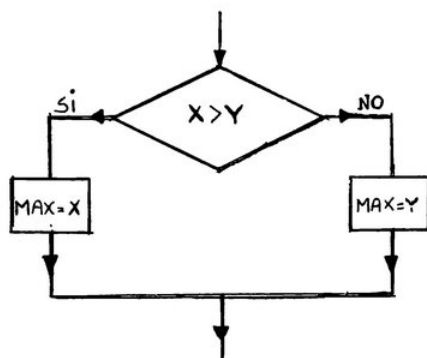
Esempio 1.

Supponiamo di avere due numeri e di voler trasferire il valore del maggiore nella variabile MAX. I passi logici che è necessario fare sono i seguenti:

a) confrontare i due numeri (chiamiamoli X e Y);

b) se X è maggiore di Y porre X nella variabile MAX;

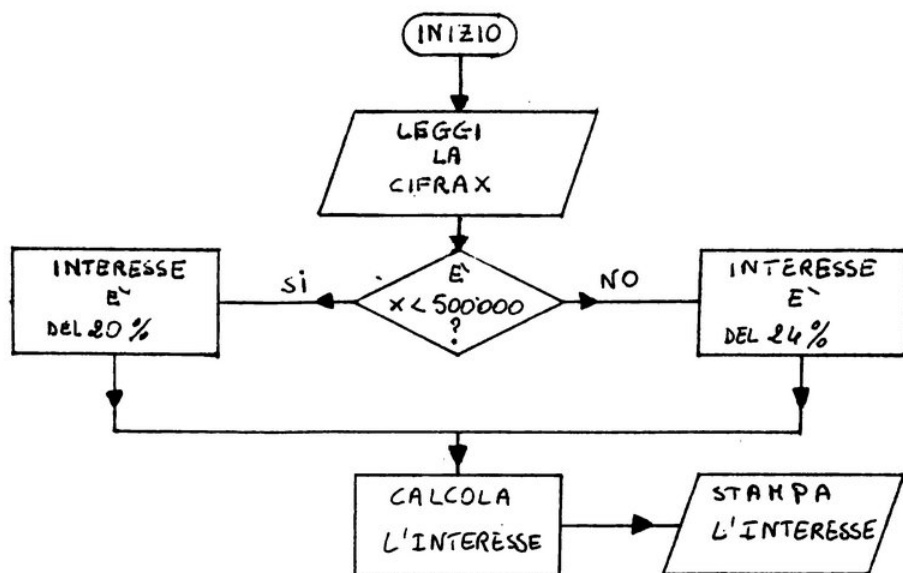
c) se X è minore di Y porre Y nella variabile MAX.

Il diagramma di flusso che segue è la rappresentazione grafica dei precedenti passi logici.

Come si può notare, il confronto tra X e Y implica una decisione (SI o NO); infatti il confronto è stato posto dentro la losanga, mentre l'assegnazione è un procedimento ed è stata posta all'interno di un rettangolo.

Il programma TI BASIC che si può dedurre da questo schema è il seguente:

```
100 IF X>=Y THEN 130
110 MAX=Y
120 GOTO 140
130 MAX=X
```

Esempio 2.

Facciamo un programma che calcoli l'interesse bancario di un prestito. Sappiamo che se la cifra è inferiore a 500.000 lire, l'interesse è del 20% altrimenti è del 24%. Ciò conduce al seguente diagramma di flusso.

Il programma che segue è una delle possibili traduzioni in TI BASIC del diagramma precedente.

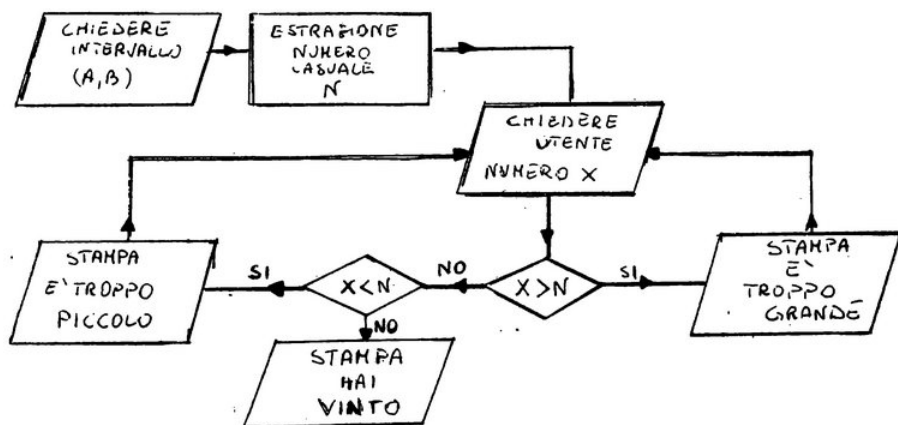
```
100 INPUT "QUAL'È LA CIFRA":X
```

```
110 IF X<500000 THEN 140
120 INT=0.2
130 GOTO 150
140 INT=0.24
150 Y=INT*X
160. PRINT "L'INTERESSE"
170 PRINT "SU";X;"È";Y
```

Come si può notare dal listato del programma, nella variabile è stato posto il valore decimale della percentuale e in Y il valore dell'interesse calcolato su X.

Esempio 3.

Mostriamo ora il diagramma di flusso di un gioco con relativo programma che è abbastanza semplice.

Si tratta di un gioco interattivo (cioè giocato contro il calcolatore) in cui bisogna indovinare un numero scelto a caso dal calcolatore. Tale numero è in un intervallo scelto dall'utente. Quando si batte il numero sulla tastiera il calcolatore lo confronta con quello da indovinare e se sono uguali stampa: "HAI INDOVINATO" altrimenti "È PIÙ GRANDE" o "È PIÙ PICCOLO" a seconda dei casi.

```
100 CALL CLEAR
110 PRINT TAB(5);"INDOVINA IL NUMERO!"::
120 INPUT "QUAL'È L'INTERVALLO?":A,B
130 N=INT (RND*(B-A+1))+A
```

```
140 CALL CLEAR
150 INPUT "QUAL'È IL NUMERO SECONDO TE?":X
160 IF X>N THEN 210
170 IF X<N THEN 240
180 CALL CLEAR
190 PRINT "BRAVO! HAI INDOVINATO"
200 STOP
210 CALL CLEAR
220 PRINT "È TROPPO GRANDE! RITENTA!"
230 GOTO 140
240 PRINT "È TROPPO PICCOLO! RITENTA!"
250 GOTO 140
```

Concludiamo questo paragrafo sottolineando che i diagrammi di flusso non sono unici: per ogni problema se ne possono costruire diversi. Quelli che abbiamo usato in questo paragrafo ci sono sembrati i più semplici per illustrare i problemi proposti.

10.2 **Documentazione di un programma: l'enunciato REM**

Si è parlato nell'introduzione della documentazione dei programmi. Documentare un programma significa introdurre in esso delle spiegazioni, dei commenti che lo rendono facilmente comprensibile a chi lo legge e che servono al programmatore stesso per ritrovare determinati punti senza dover scorrere l'intero programma. Ciò si può ottenere con l'enunciato REM (Remark: commento).

Esso è così strutturato: N. di linea REM frase di commento. Per lo scopo a cui sono destinate le osservazioni contenute nel comando REM, non è necessario che il calcolatore le legga ed infatti quando trova scritto REM salta immediatamente alla istruzione successiva.

In un programma possono essere inseriti in qualsiasi punto innumerevoli commenti. Vediamo un esempio:

```
100 REM AREA E CIRCONFERENZA DI UN CERCHIO
110 INPUT "DAMMI IL RAGGIO:":R
120 A=3.14*R^2
130 C=2*3.14*R
140 REM STAMPA DELL'AREA E DELLA CIRCONFERENZA
150 PRINT "AREA=";A::"CIRCONF.=";C
160 STOP
```

È evidente che gli enunciati REM forniscono al programmatore un utile mezzo per spiegare il programma. Se circondiamo con righe bianche l'enunciato REM, le osservazioni in esso contenute risultano distinte dal resto del programma e costituiscono una più chiara presentazione di quest'ultimo.

10.3 **Informazioni sulla memoria**

È comodo, soprattutto per la stesura di programmi molto lunghi, cioè che occupano una grande quantità di memoria, poter ottenere dal calcolatore informazioni sulla quantità di memoria ancora disponibile. In TI BASIC non esistono funzioni dirette che possano dare questa informazione (in EXTENDED BASIC esiste per far ciò la funzione SIZE).

Un modo, però, per raggiungere comunque questo scopo è quello di introdurre una speciale subroutine all'inizio del programma attualmente in memoria. Essa è una subroutine temporanea che deve essere cancellata appena ottenuta l'informazione sulla memoria, ed è la seguente:

```
1 ML=ML+8
2 GOSUB 1
```

Abbiamo usato i numeri di linea 1 e 2 per evidenziare che detta subroutine deve essere posta sempre all'inizio dei programmi già battuti. Se ora facciamo girare il programma così modificato, battendo RUN, il calcolatore, dopo qualche secondo, arresterà l'esecuzione stampando sul video il messaggio:

MEMORY FULL IN 1

A questo punto diamo il comando diretto PRINT ML ed il numero che compare sullo schermo corrisponde al numero di byte di memoria ancora disponibili. Se la suddetta subroutine viene introdotta quando in memoria non c'è alcun programma si ottiene: 14536.

10.4 **Confronti multipli: gli operatori AND e OR**

In alcuni linguaggi come il FORTRAN e l'EXTENDED BASIC c'è la possibilità di fare più confronti contemporaneamente usando gli operatori AND e OR. In TI BASIC l'uso di questi operatori non è previsto. Quando, infatti, si devono confrontare più variabili (o più espressioni) tra di loro, bisogna usare un'istruzione per ogni confronto e nel programma si ha una più o meno lunga serie di IF-THEN. Ad esempio volendo controllare che la variabile A sia com-

presa nell'intervallo (1,5), cioè sia maggiore di 1 e minore di 5, dovremmo scrivere così:

```
100 IF A<=1 THEN 200
110 IF A>=5 THEN 200
120 PRINT "A È NELL'INTERVALLO"
130 ..........

200 PRINT "È FUORI INTERVALLO"
```

Esiste però un modo per superare questo ostacolo. Se i confronti vengono racchiusi tra parentesi, i simboli *e + fungono rispettivamente da operatori AND e OR. Apriamo qui una breve parentesi per chi non conosce questi operatori e non sa, quindi, qual'è la loro funzione.

Si definisce proposizione logica una espressione dove compaiono due variabili (o espressioni algebriche) collegate da uno qualunque degli operatori relazionali (vedi paragrafo 3.4). A tale espressione logica viene assegnato il valore «vero» se è verificata, «falso» se non lo è. Vediamo degli esempi: se A=5 B=3 C=1 allora si ha:

A<B	"falsa"
A>B	"vera"
A+C<>B+A	"vera"
A∧2 <=B*C	"falsa"

AND e OR sono due operatori logici che possono collegare, al fine di fare più confronti contemporaneamente, le proposizioni suddette.

Riprendiamo l'esempio fatto all'inizio dove si controllava la variabile A. Usando l'operatore logico AND, esso può essere così riscritto:

```
100 IF (A<=1)*(A<=5) THEN 200
110 PRINT "A È FUORI INTERVALLO"

200 PRINT "A È NELL'INTERVALLO
```

Si può, comunque, usare anche l'operatore OR e in tal caso è:

```
100 IF (A<=1)+(A>=5) THEN 200
110 PRINT "A È DENTRO L'INTERVALLO"


200 PRINT "A È FUORI INTERVALLO"
```

Come si può capire dagli esempi precedenti, l'operatore OR (+) è verificato se almeno una delle espressioni logiche è verificata, e l'operatore AND (*) è verificato se tutte le espressioni logiche sono verificate.

Si possono collegare anche più di due espressioni logiche usando però sequenzialmente sempre lo stesso operatore.

Ad esempio:

```
100 IF (A<>1)*(A<>3)*(A<>-7) THEN 200


200.
```

Tale IF-THEN sarà verificato se il numero A è diverso contemporaneamente da 1, da 3, e da -7 (A=0 ne è un esempio).

10.5 **Una routine per la stampa comandata**

Quando vogliamo stampare un'informazione sul video, dobbiamo servirci del comando PRINT che, come sappiamo, effettua la stampa a partire dall'ultima riga in basso. Poi, quando il cursore completa la scrittura del messaggio, avviene lo scrolling verso l'alto di tutto ciò che è presente sullo schermo. Questa caratteristica dell'istruzione PRINT diviene un inconveniente quando, lavorando con la grafica per esempio, vogliamo eseguire la stampa di un informazione senza provocare lo scrolling oppure la vogliamo eseguire su una ben precisa zona dello schermo.

Esiste, però, una routine che permette di ovviare a questi inconvenienti. Essa può essere implementata su un programma (sotto forma di subroutine definita dall'utente) e richiamata quando occorre.

Per fissare meglio le idee, supponiamo che l'informazione da stampare sia: BRAVO 100 PUNTI, e che si voglia stampare il primo carattere (cioè B) nella posizione (10,7) cioè nella riga 10 colonna 7, e tutti gli altri caratteri di

seguito a B.

Vediamo come tale sottoprogramma può essere inserito in un programma generale:

```
100 Z$="BRAVO 100 PUNTI"
110 R=10
120 C=7
130 GOSUB 500



500 FOR J=1 TO LEN (Z$)
510 A=ASC(SEG$(Z$,J,1))
520 CALL HCHAR(R,C,A)
530 C=C+1
540 NEXT J
550 RETURN
```

Come si può notare, il funzionamento di tale routine si fonda essenzialmente sull'uso della istruzione SEG$. Quando il programma va alla linea 500, con l'istruzione LEN (Z$) si assegna al valore finale della variabile corrente J il numero corrispondente alla lunghezza della stringa ed inizia il ciclo.

Il calcolatore assegna alla variabile corrente J il valore 1 e quindi, alla linea 510, legge il codice ASCII del primo carattere della variabile Z$. La linea 520 stampa sullo schermo, nella posizione voluta, tale carattere e quindi (540) viene incrementato il numero di colonna (C) per la stampa del prossimo carattere. Il procedimento, sin qui descritto, si ripete tante volte fino al completamento del ciclo.

10.6 **Algoritmi matematici**

Noi siamo convinti che queste pagine non servano solo a coloro che sono interessati a problemi scientifici ma siano utili a tutti quelli che si interessano di programmazione e la applicano nei più svariati campi. Infatti molti problemi (salvo casi eccezionali) possono essere opportunamente ricondotti a problemi matematici la cui risoluzione è certamente più semplice da schematizzare dato l'alto rigore e l'assenza di ambiguità nel linguaggio matematico.

Ordinare una lista di nomi, per esempio, in ordine alfabetico non è un'applicazione matematica ma il suo algoritmo discende direttamente da quello usato per ordinare un vettore (ossia una lista numerica).

Noi consigliamo all'utente di studiare e capire il maggior numero di algoritmi già esistenti prima di cominciare a costruirne di propri; è in questo modo che ci si crea la logica necessaria per affrontare ogni problema.

Data la semplicità dei programmi che abbiamo scelto di trattare, omettiamo i diagrammi di flusso presentando soltanto i listati TI BASIC degli algoritmi con relative spiegazioni.

10.6.1 **Algoritmo che verifica se un numero è primo**

Ricordiamo innanzitutto che cosa è un numero primo: un numero si dice primo se è divisibile solo per se stesso e per l'unità. In questo algoritmo, comunque, oltre che della definizione, si fa uso sia della considerazione che qualsiasi numero pari non è primo poichè divisibile almeno per 2, sia di un teorema che afferma l'inutilità di provare divisori superiori alla radice del numero stesso e che qui non dimostriamo.

```
100 REM
110 REM PROGRAMMA CHE CALCOLA SE UN NUMERO È PRIMO
120 REM
130 INPUT "DAMMI IL NUMERO":A
140 IF A/2=INT(A/2) THEN 200
150 FOR I=3 TO INT(SQR(A))+1 STEP 2
160 IF A/I=INT(A/I) THEN 200
170 NEXT I
180 PRINT "IL NUMERO È PRIMO"
190 STOP
200 PRINT "IL NUMERO NON È PRIMO"
210 END
```

10.6.2 **Risoluzione di un'equazione di 2° grado**

Vogliamo qui proporre un programma che calcola le radici di un'equazione di 2° grado nel modo più generale possibile, cioè anche quando le soluzioni sono complesse e coniugate. Ci riferiamo ad un'equazione generale nella forma:

$ax^2 + bx + c = 0$

Sappiamo che le soluzioni sono:

$$x_1 = \frac{-b + \sqrt{b^2-4ac}}{2a} \quad e \quad x_2 = \frac{-b- \sqrt{b^2-4ac}}{2a}$$

se b^2-4ac è maggiore o uguale a zero; altrimenti sono:

$$x_1 = \frac{-b}{2a} + \frac{i\sqrt{4ac-b^2}}{2a} \quad e \quad x_2 = \frac{-b}{2a} - \frac{i\sqrt{4ac-b^2}}{2a}$$

dove con «i» si è indicata l'unità immaginaria, cioè $\sqrt{-1}$. Vediamo quindi il listato.

```
100 INPUT "DAMMI I COEFFICIENTI A,B,C": A,B,C
110 REM
120 REM CALCOLO DEL SEGNO DEL DISCRIMINANTE
130 DISCR=B∧2-4*A*C
140 IF DISCR<0 THEN 230
150 X1=(-B+SQR(DISCR))/(2*A)
160 X2=(-B-SQR(DISCR))/(2*A)
170 CALL CLEAR
180 PRINT "LE SOLUZIONI SONO REALI"
190 PRINT: "X1=";X1::"X2=";X2
200 STOP
210 REM INDICHIAMO CON REX LA PARTE REALE
220 REM CHE COINCIDE PER ENTRAMBE LE SOLUZIONI
230 REX=-B/(2+A)
240 IMMX1=SQR(-DISCR)/(2+A)
250 IMMX2=-SQR(-DISCR)/(2+A)
260 CALL CLEAR
270 PRINT "LE SOLUZIONI SONO COMPLESSE E CONIUGATE"
280 PRINT: "X1=";REX;"+i";IMMX1::"X2=";REX;"-i";IMMX2
290 END
```

Concludiamo questo programma con un'osservazione sulle linee 240 e 250. Sappiamo che alcune funzioni di libreria non ammettono argomento negativo: una di queste è proprio la funzione SQR. Perciò si è reso necessario, nelle suddette linee, moltiplicare la variabile DISCR per -1 cioè cambiarla di segno. Per arrivare infatti con il programma alle linee in questione, è necessario che si verifichi l'IF-THEN della linea 140 (e quindi che DISCR sia negativa).

10.6.3 **Ordinamento di un vettore**

Consideriamo il problema dell'ordinamento in modo crescente di una lista di N numeri dati. Questo algoritmo è chiamato di sort (sort: ordinamento).

La procedura per l'ordinamento consiste nell'esaminare tutta la lista alla ricerca del numero più piccolo per scambiarlo poi con il primo della lista. Restano quindi N-1 elementi ai quali si riapplica lo stesso ragionamento scambiando, quindi, il più piccolo fra essi con il secondo elemento della lista. Si passa poi ai rimanenti N-2 elementi procedendo analogamente fino all'ultimo della lista.

Il numero dei passaggi che in tal modo si compiono è ovviamente N-1.

Per trovare, ad ogni passaggio, il numero più piccolo, si confronta il primo elemento con il successivo e se questo è più grande si scambia altrimenti no.

Per effettuare lo scambio si procede in questo modo: se V(3) e V(4) sono gli elementi da scambiare si eseguono le seguenti istruzioni:

```
200 TEMP=V(3)
210 V(3)=V(4)
220 V(4)=TEMP
```

questo perché se ponessimo direttamente V(3) in V(4) e poi V(4) in V(3), alla fine ci troveremmo con l'avere nelle variabili V(3) e V(4) lo stesso valore (cioè V(3)). Infatti quando poniamo V(3) in V(4), avviene che il contenuto della locazione in cui è contenuto V(4) viene perso ed il suo posto è preso da V(3); e quindi quando andiamo a mettere V(4) in V(3), non facciamo nient'altro che riconfermare lo stesso contenuto. Per risolvere questo problema viene temporaneamente salvato il contenuto della locazione V(3) in una variabile temporanea (TEMP) e solo allora, in V(3), viene scritto il contenuto di V(4). Infine, il contenuto di V(3) (che ora è in TEMP), viene depositato nella locazione V(4) (vedi linea 220).

Vediamo ora il programma completo che ordina in senso crescente una lista o vettore) di N elementi.

```
100 REM
110 REM ORDINAMENTO DI UN VETTORE
120 REM
130 DIM V(100)
140 INPUT "QUANTI ELEMENTI DEVO ORDINARE (MAX.100)":N
150 FOR I=1 TO N
160 PRINT "DAMMI L'ELEMENTO";I
170 INPUT V(I)
```

```
180 REM
190 REM INGRESSO DATI TRAMITE TASTIERA
200 REM
210 NEXT I
220 FOR I=1 TO N-1
230 FOR J=1+1 TO N
240 IF V(J)>=V(I) THEN 280
250 REM PROCEDURA DI SCAMBIO
255 TEMP=V(J)
260 V(J)=V(I)
270 V(I)=TEMP
280 NEXT J
290 NEXT I
300 REM
310 REM STAMPA DEL VETTORE ORDINATO
320 REM
330 FOR L=1 TO N
340 PRINT V(L)
350 NEXT L
360 END
```

Osserviamo che l'impostazione di richiesta dei dati è di tipo generale. Ciò vuol dire che i suddetti non risiedono permanentemente nel programma (ad esempio tramite un blocco DATA), ma sono di volta in volta assegnati al vettore V tramite l'istruzione INPUT Si tratta quindi di dati «dinamici» e non «statici».

10.6.4 **Un programma di statistica**

Presentiamo in questo paragrafo un programma che produce la media e la deviazione standard di N elementi. Tali elementi possono essere ad esempio le vendite effettuate da un magazzino nel corso di un anno, oppure le misure eseguite su un particolare campione, ecc. Vediamo quali sono le formule usate in questo programma: siano x_1, x_2, $x_3 \ldots x_i \ldots$, x_n gli N elementi. Definiamo media della successione o semplicemente media aritmetica la quantità seguente:

$$X_M = \frac{x_1 + x_2 + x_3 \ldots\ldots + x_n}{N}$$

dove X_M è appunto la media e N è il numero degli elementi; definiamo deviazione dalla media di x_i la quantità:

$d_i = (x_i - x)$ (per i = 1,2,3,.....,n-1,n)

e deviazione standard (e la indichiamo con S):

$$S = \frac{d_1^2 + d_2^2 + d_3^2 + \ldots\ldots + d_n^2}{N}$$

Nel programma che segue, mettiamo gli elementi in un array monodimensionale per maggior comodità.

Sia in questo che nel programma del paragrafo 10.5.3, la condizione di un numero massimo di 100 elementi può essere rimossa modificando la linea dichiarativa del DIM. Ad esempio, se volessimo trattare un numero di 1000 elementi, sarà sufficiente scrivere:

130 DIM V(1000)   (oppure: 130 DIM X(1000))

```
100 REM
110 REM APPLICAZIONE DI STATISTICA
120 REM
130 DIM X(100)
140 CALL CLEAR
150 INPUT "QUANTI ELEMENTI (MAX.100)":N
155 FOR I=1 TO N
160 PRINT "DAMMI L'ELEMENTO";I
170 INPUT X(I)
180 NEXT I
190 REM
200 REM CALCOLO DELLA MEDIA
210 REM
220 FOR I=1 TO N
230 SOM=SOM+X(I)
240 NEXT I
250 MED=SOM/N
260 REM
270 REM CALCOLO DELLA DEVIAZIONE STANDARD
280 REM
290 FOR I=1 TO N
300 DEV=(X(I)-MED)  2+DEV
310 NEXT I
320 S=SQR(DEV/N)
330 REM
```

```
340 REM STAMPA DEI RISULTATI
350 REM
360 CALL CLEAR
370 PRINT "LA MEDIA ARITMETICA È"::"XM";MED
380 PRINT: "LA DEVIAZIONE STANDARD È"::"S=";S
390 END
```

10.6.5 **Calcolo dell'integrale definito di una funzione**

Vediamo infine un programma che calcola il valore dell'integrale di una funzione f(x) nell'intervallo (a,b).

Il teorema della media per gli integrali ci dice che l'integrale cercato è uguale a $(b\text{-}a)y_1$ dove (b-a) è la lunghezza dell'intervallo ed y_1 è la media di tutti i valori f(x) che la funzione assume nel suddetto intervallo. Poiché tali valori sono infiniti, scegliamo un campione casuale di valori di x in (a,b) e per questi calcoliamo i relativi f(x). Quindi facciamo la somma dei valori così ottenuti e poi dividiamo per il numero di elementi formante il campione.

Naturalmente più il campione contiene elementi e più il valore dell'integrale sarà meglio approssimato; ma ciò implica un aumento del tempo-macchina necessario per l'esecuzione del programma. Sceglieremo allora un campione che sia il prodotto di un ragionevole compromesso tra la precisione voluta e il tempo d'esecuzione richiesto.

Prima di passare al programma è d'obbligo una naturale precisazione. La funzione di cui vogliamo conoscere l'integrale, deve essere definita ogni volta alla linea 170. Ciò implica il fatto di dover richiamare ogni volta tale linea per inserire la funzione desiderata.

Nel programma che segue, la funzione che consideriamo è la nota parabola $y=x^2$.

```
100 REM
110 REM PROGRAMMA PER IL CALCOLO DI UN INTEGRALE
120 REM
130 CALL CLEAR
135 REM RICHIESTA DELL'INTERVALLO D'INTEGRAZIONE (A,B)
140 REM
150 INPUT "DAMMI L'INTERVALLO (A,B)":A,B
155 REM
160 REM DEFINIZIONE DELLA FUNZIONE INTEGRANDA
165 REM
170 DEF F(X)=X^2
```

```
180 N = B-A
190 FOR I = 1 TO 1000
200 Z = A + RND * N
210 SUM = SUM + F(Z)
220 NEXT I
230 REM
240 REM CALCOLO DEL VALOR MEDIO DI F(X)
250 REM
260 MED = SUM/1000
270 REM
280 REM CALCOLO DELL'INTEGRALE
290 REM
300 INTEG = MED * N
310 CALL CLEAR
320 PRINT "IL VALORE DELL'INTEGRALE È";INTEG
330 END
```

Come si può osservare il numero di campioni scelti in questo caso è 1000 (vedi linea 190).

Concludiamo questo paragrafo avvertendo che l'algoritmo appena esposto, a differenza di altri, risulterà difficile da comprendere se non addirittura astruso, per coloro che non hanno conoscenze di analisi matematica. Chi si trova quindi in questa condizione può direttamente passare oltre.

ESERCIZI PROPOSTI (CAP.10)

Non presentiamo in questo capitolo, come per tutti gli altri del libro, la serie di esercizi risolti in quanto è il capitolo stesso che ne ha fornito continuamente gli esempi

1) In riferimento agli algoritmi matematici esposti in precedenza, cercare di rappresentare i relativi diagrammi di flusso. (N.B.: questa operazione vuole solo avere lo spirito di una esercitazione ma è ovvio che nella progettazione di un programma, il diagramma di flusso precede sempre il listato).

2) Qual'è lo scopo dell'enunciato REM all'interno di un programma? È un'istruzione che il calcolatore «esegue»? Spiegare.

3) Realizzare un programma che, sfruttando l'algoritmo di sort (quello per l'ordinamento di un vettore), ordini una lista di nomi di persona per ordine alfabetico. (Consiglio: sfruttare la funzione ASC e ricavare, dal vettore alfanumerico, il relativo vettore numerico; quindi ordinarlo secondo la tecnica già vista e...)

4) Disegnare il diagramma di flusso e compilare un programma che calcola:

a) la media tra 10 voti riportati da 20 studenti di una classe;

b) la media della classe attraverso le medie individuali;

c) la differenza tra le medie individuali e la media della classe.

(Consiglio: richiamarsi all'algoritmo illustrato nel sotto-paragrafo 10.5.4).

5) Quando si chiede un prestito ad una banca, la restituzione viene suddivisa, in genere, in rate mensili tutte eguali che vengono pagate per un determinato numero di anni.

Le rate sono costituite da una parte che serve a pagare l'interesse mensile, e da un'altra che serve a ridurre l'ammontare del debito.

All'inizio la maggior parte della rata serve a pagare l'interesse, ma mano che si va avanti nei pagamenti l'interesse mensile diminuisce e aumenta la parte che va a ridurre l'ammontare del debito.

Scrivere un programma BASIC, con relativo diagramma di flusso, in cui, conoscendo l'ammontare del prestito, il tasso d'interesse mensile ed il numero di anni con cui si intende pagare il debito, si calcola il valore della rata mensile. Esso si ottiene dalla formula:

$$R = IP\left(\frac{(1+I)^{12a}}{(1+I)^{12a}-1}\right)$$

dove:
R = rata mensile in lire
P = ammontare del prestito in lire
I = rata d'interesse mensile espressa in decimali per esempio se l'interesse è del 2%, I = 0,02
12a = numero totale di anni espresso in mesi.
(Consiglio: per fare un programma generale, introducete i parametri P,I e a con degli INPUT).

6) Fare un programma TI BASIC che scomponga un numero in fattori primi, usando opportunamente l'algoritmo del sottoparagrafo 10.5.1.

CAPITOLO 11

Introduzione ai giochi con il computer

Abbiamo voluto dedicare un intero capitolo ai giochi che si possono realizzare con il calcolatore principalmente per due motivi:

1) gli utenti di home-computer (e quindi anche del TI 99/4A) iniziano quest'hobby quasi sempre con il solo scopo di giocare con e/o contro il calcolatore (cosa tra l'altro che non deve suonare come un rimprovero: infatti ci giochiamo anche noi!);

2) per imparare ad usare buone tecniche di programmazione i giochi sono spesso un piacevole esercizio. Infatti, nella progettazione del programma, si devono spesso affontare problemi di grafica (non sempre banali) e spesso anche di strategia (vedi Othello, dama, scacchi, ecc.) che quasi sempre impongono una complessa analisi dell'algoritmo di risoluzione.

Naturalmente gli esempi che abbiamo scelto di trattare non sono particolarmente complicati e soprattutto non richiedono la progettazione di una raffinata strategia. Ciò perché il nostro pricipale obiettivo è quello di rendere più chiari possibile tutti i nostri esempi. Vogliamo, cioè, che l'utente possa trarre da quello che «vorremmo aver seminato in questo libro», le basi necessarie per affrontare in maniera autosufficiente l'affascinante mondo della programmazione.
Quindi non riteniamo giusto cominciare ad insegnare direttamente le cose più difficili lasciando da parte quelle più facili.

11.1 Il gioco dei dadi

Questo gioco, come tutti sanno, è completamente affidato al caso e quindi di facile realizzazione non dovendo tener conto, nella programmazione, neanche della più elementare strategia.
Per l'estrazione casuale del numero viene usata, ovviamente, la funzione RND abbinata all'enunciato RANDOMIZE (quest'ultimo per avere, ogni volta che il programma gira, sequenze casuali diverse).
Vediamo ora il diagramma di flusso e subito dopo il programma TI BASIC.

INIZIO
DEFINIZIONE GRAFICA FACCE
STAMPA DI PRESENTAZIONE
D
A
ESTRAZIONE DI UN NUMERO CASUALE
IL NUMERO A QUALE FACCIA CORRISPONDE ?
SUBROUTINE FACCIA 2
SUBROUTINE FACCIA 5
SUBROUTINE FACCIA 1
SUBROUTINE FACCIA 3
SUBROUTINE FACCIA 4
SUBROUTINE FACCIA 6
HA ESTRATTO DUE NUMERI IL GIOCATORE ?
NO
SI
STAMPA LA LORO SOMMA
B
C

C
INSIEME DELLE PROCEDURE PER L'ESTRA=
ZIONE DI DUE NUMERI PER IL CALCOLATORE.
IDENTICO A BLOCCO PRECEDENTE DA A a B.
SOMMA DEL
GIOCATORE MAGGIORE
QUELLA CALCOLATORE
?
NO
SI
STAMPA
HO VINTO!
STAMPA
HAI VINTO!
STAMPA
VUOI GIOCARE
ANCORA?
RISPOSTA
SI
D
NO
CIAO CIAO
FINE

```
100 REM
110 REM
120 REM DEFINIZ. FACCIA 6
130 REM
140 REM
150 CALL CHAR(130,"FF80809898808098")
160 CALL CHAR(131,"FF01011919010119")
170 CALL CHAR(132,"98808098988080FF")
180 CALL CHAR(133,"19010119190101FF")
190 REM
200 REM
210 REM DEFINIZ. FACCIA 5
220 REM
230 REM
240 CALL CHAR(134,"FF80809898808081")
250 CALL CHAR(135,"FF01011919010181")
260 CALL CHAR(136,"81808098988080FF")
270 CALL CHAR(137,"81010119190101FF")
280 REM
290 REM
300 REM DEFINIZ. FACCIA 4
310 REM
320 REM
330 CALL CHAR(138,"FF80809898808080")
340 CALL CHAR(139,"FF01011919010101")
350 CALL CHAR(140,"80808098988080FF")
360 CALL CHAR(141,"01010119190101FF")
370 REM
380 REM
390 REM DEFINIZ. FACCIA 3
400 REM
410 REM
420 CALL CHAR(142,"FF80808080808081")
430 CALL CHAR(143,"81010101010101FF")
440 REM
450 REM
460 REM DEFINIZ. FACCIA 2
470 REM
480 REM
490 CALL CHAR(144,"FF80808080808098")
500 CALL CHAR(145,"FF01010101010119")
510 CALL CHAR(146,"98808080808080FF")
520 CALL CHAR(147,"19010101010101FF")
```

```
530 REM
540 REM
550 REM DEFINIZ. FACCIA 1
560 REM
570 REM
580 CALL CHAR(148,"FF80808080808081")
590 CALL CHAR(149,"FF01010101010181")
600 CALL CHAR(150,"81808080808080FF")
610 CALL CHAR(151,"81010101010101FF")
620 REM
630 REM
640 REM PRESENTAZIONE
650 REM
660 REM
670 CALL CLEAR
680 PRINT TAB(4); "DADI CON IL CALCOLATORE"::::::::
690 FOR T=1 TO 1000
700 NEXT T
710 CALL CLEAR
720 PRINT "QUANDO SEI PRONTO BATTI LA"
730 PRINT: "BARRA SPAZIATRICE PER TIRARE"
740 PRINT: "I TUOI DADI"
750 CALL KEY (0,K,S)
760 IF S=0 THEN 750
770 IF K=31 THEN 750
780 CALL CLEAR
790 REM
800 REM
810 REM ESTRAZ. DADI UTENTE
820 REM
830 REM
840 RANDOMIZE
850 P1=INT(RND*6)+1
860 Z=0
870 ON P1 GOSUB 1180,1240,1300,1360,1420,1480
880 P2=INT(RND*6)+1
890 Z=4
900 ON P2 GOSUB 1180,1240,1300,1360,1420,1480
910 PRINT TAB(4);"TU HAI FATTO";P1+P2;"PUNTI"::
920 PRINT: "ORA TOCCA A ME!!!!"::::::::::
930 FOR T=1 TO 2000
940 NEXT T
950 REM
```

```
 960 REM
 970 REM ESTRAZ. DADI CALCOL.
 980 REM
 990 REM
1000 P3=INT(RND*6)+1)
1010 Z=0
1020 ON P3 GOSUB 1180,1240,1300,1360,1420,1480
1030 P4=INT(RND*6)+1
1040 Z=4
1050 ON P4 GOSUB 1180,1240,1300,1360,1420,1480
1060 PRINT "IO, INVECE, HO FATTO"; P3+P4;:"PUNTI!"
1070 FOR T=1 TO 2000
1080 NEXT T
1090 CALL CLEAR
1100 GOSUB 1640
1110 PRINT "VUOI PROVARCI ANCORA?"::::::::::
1120 INPUT A$
1130 IF A$="SI" THEN 710
1140 CALL CLEAR
1150 PRINT "PIACERE DI AVER GIOCATO CON"
1160 PRINT: "TE! CIAO CIAO!!!"
1170 STOP
1180 A1=148
1190 A2=149
1200 A3=150
1210 A4=151
1220 GOSUB 1590
1230 RETURN
1240 A1=144
1250 A2=145
1260 A3=146
1270 A4=147
1280 GOSUB 1590
1290 RETURN
1300 A1=142
1310 A2=135
1320 A3=136
1330 A4=143
1340 GOSUB 1590
1350 RETURN
1360 A1=138
1370 A2=139
1380 A3=140
```

```
1390 A4 = 141
1400 GOSUB 1590
1410 RETURN
1420 A1 = 134
1430 A2 = 135
1440 A3 = 136
1450 A4 = 137
1460 GOSUB 1590
1470 RETURN
1480 A1 = 130
1490 A2 = 131
1500 A3 = 132
1510 A4 = 133
1520 GOSUB 1590
1530 RETURN
1540 REM
1550 REM
1560 REM SOTTOPROGRAMMA PER IL DISEGNO DEI DADI
1570 REM
1580 REM
1590 CALL HCHAR(18,14+Z,A1)
1600 CALL HCHAR(18,15+Z,A2)
1610 CALL HCHAR(19,14+Z,A3)
1620 CALL HCHAR(19,15+Z,A4)
1630 RETURN
1640 IF P1+P2>P3+P4 THEN 1700
1650 CALL CLEAR
1660 PRINT "MI DISPIACE MA HO VINTO IO!!"::::::::::::
1670 FOR T=1 TO 1000
1680 NEXT T
1690 RETURN
1700 CALL CLEAR
1710 PRINT "HAI VINTO TU! CHE FORTUNA!!"::::::::::::
1720 FOR T=1 TO 1000
1730 NEXT T
1740 RETURN
1750 END
```

Abbiamo volutamente evitato, in questo gioco ed in quelli che seguiranno, di dare molta rilevanza alla presentazione estetica, affinché l'utente afferri l'essenza degli algoritmi proposti senza farsi fuorviare dagli elementi di contorno. Essa d'altro canto è di facile realizzazione ed è lasciata alla fantasia del programmatore.

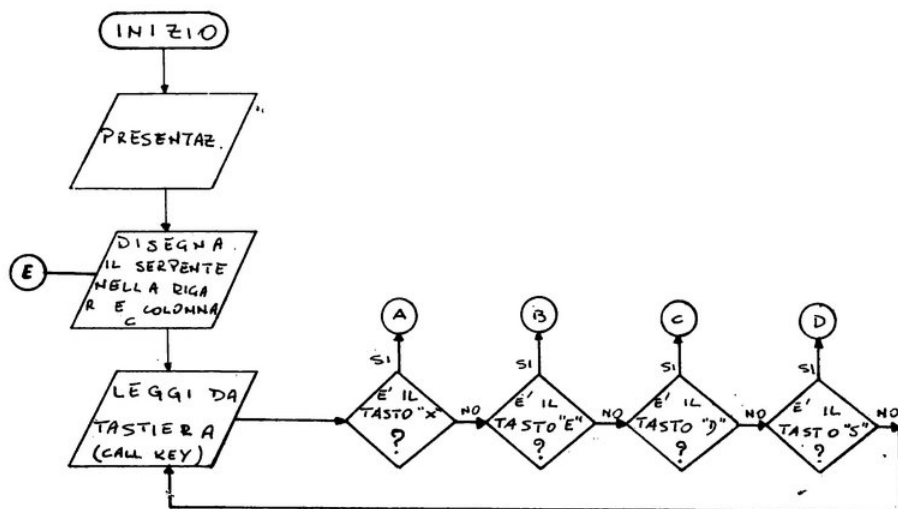
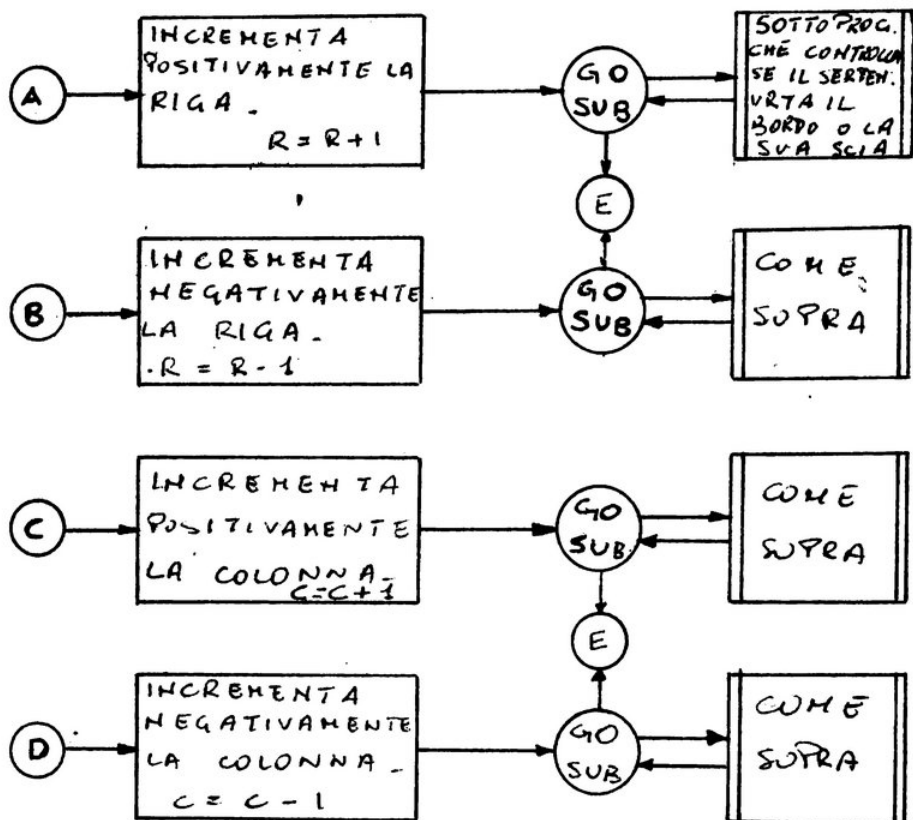
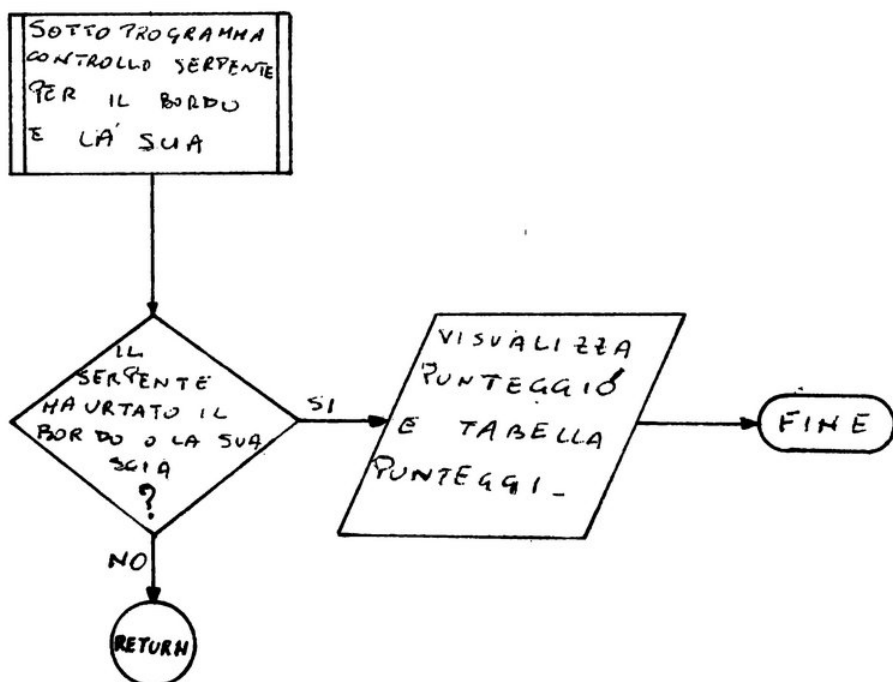
```
740 IF U2=D1 THEN 1090
750 IF U2=U1 THEN 1130
760 GOTO 1070
770 IF D2=C1 THEN 850
780 IF D2=D1 THEN 820
790 IF U2=C1 THEN 1090
800 IF U2=D1 THEN 1090
810 GOTO 1070
820 IF U2=C1 THEN 1190
830 IF U2=D1 THEN 1190
840 GOTO 1130
850 IF U2=D1 THEN 1150
860 GOTO 1090
870 IF D2=C1 THEN 940
880 IF D2=U1 THEN 920
890 IF U2=C1 THEN 1090
900 IF U2=U1 THEN 1130
910 GOTO 1070
920 IF U2=C1 THEN 1150
930 GOTO 1090
940 IF U2=U1 THEN 1190
950 GOTO 1090
960 IF D2=D1 THEN 1030
970 IF D2=U1 THEN 1010
980 IF U2=D1 THEN 1130
990 IF U2=U1 THEN 1170
1000 GOTO 1110
1010 IF U2=D1 THEN 1190
1020 GOTO 1130
1030 IF U2=U1 THEN 1050
1040 GOTO 1170
1050 PRINT TAB(7);"HAI INDOVINATO!"
1060 STOP
1070 PRINT "0"
1080 GOTO 240
1090 PRINT "00"
1100 GOTO 240
1110 PRINT "1"
1120 GOTO 240
1130 PRINT "10"
1140 GOTO 240
1150 PRINT "000"
1160 GOTO 240
```

```
1170 PRINT "11"
1180 GOTO 240
1190 PRINT "100"
1200 GOTO 240
1210 RETURN
1220 END
```

A questo punto del libro speriamo che l'utente sia in grado, attraverso il listato ed il diagramma di flusso, di comprendere la logica di funzionamento del programma senza bisogno di ulteriori spiegazioni.

11.3 Il gioco del serpente

Il gioco che segue è stato appositamente non documentato per farvi notare come sia più difficile rendersi conto dei vari passaggi logici senza alcun commento (REM). Con l'aiuto del diagramma di fusso e del listato, provate a documentarlo voi.

A
INCREMENTA POSITIVAMENTE LA RIGA.
R = R + 1
GO SUB
SOTTOPROG. CHE CONTROLLA SE IL SERPEN. URTA IL BORDO O LA SUA SCIA
E
B
INCREMENTA NEGATIVAMENTE LA RIGA.
R = R - 1
GO SUB
COME SOPRA
C
INCREMENTA POSITIVAMENTE LA COLONNA.
C=C+1
GO SUB
COME SOPRA
E
D
INCREMENTA NEGATIVAMENTE LA COLONNA.
C = C - 1
GO SUB
COME SOPRA

```
100 RANDOMIZE
110 CALL SCREEN (16)
120 CALL CLEAR
130 PRINT "IL GIOCO DEL SERPENTE!!!!!!":::::::::
140 FOR T=1 TO 1000
150 NEXT T
160 CALL CLEAR
170 PRINT "CIAO! QUAL'È IL TUO NOME?"
180 INPUT B$
190 C$=SEG$(B$,LEN(B$),1)
200 CALL CLEAR
210 PRINT "APPENA SEI PRONTO BATTI UN"::"TASTO QUALUNQUE
    E... BUONA"
220 PRINT: "FORTUNA!!!!!!":::::
220 CALL KEY(0,K,S)
230 IF S=0 THEN 220
```

```
240 CALL CLEAR
250 CALL CHAR(130,"AA55AA55AA55AA55")
260 CALL HCHAR(24,1,130,64)
270 CALL VCHAR(1,32,130,72)
280 R=INT(RND*23)
290 C=INT(RND*30)+3
300 X=0
310 T=0
320 CALL HCHAR(R,C,30)
330 X=X+1
340 CALL KEY(0,K,S)
350 T=T+1
360 IF S=0 THEN 340
370 IF K=83 THEN 410
380 IF K=68 THEN 440
390 IF K=69 THEN 470
400 IF K=88 THEN 500
405 GOTO 340
410 C=C-1
420 GOSUB 530
430 GOTO 320
440 C=C+1
450 GOSUB 530
460 GOTO 320
470 R=R-1
480 GOSUB 530
490 GOTO 320
500 R=R+1
510 GOSUB 530
520 GOTO 320
530 CALL GCHAR(R,C,Y)
540 IF (Y=130)+(Y=30) THEN 560
550 RETURN
560 GOSUB 580
570 GOTO 640
580 CALL HCHAR (R,C,30)
590 FOR I=1 TO 16
600 CALL SCREEN (I)
610 CALL SOUND(500,-1,1)
620 NEXT I
630 RETURN
640 CALL CLEAR
650 PRINT "MI DISPIACE MA HAI URTATO!!!"
```

```
660 PRINT:"COMUNQUE IL TUO PUNTEGGIO È"::INT(X*X/T)::::::
670 FOR T=1 TO 2000
680 NEXT T
690 PRINT TAB(8);"TABELLA PUNTEGGI"
700 PRINT::"SOVRUMANO";TAB(20);"OLTRE 500"
710 PRINT:"ECCEZIONALE";TAB(20);"OLTRE 400"
720 PRINT:"BRAVISSIMO";TAB(20);"OLTRE 300"
730 PRINT:"BRAVO";TAB(20);"OLTRE 200"
740 PRINT:"DISCRETO";TAB(20);"OLTRE 150"
750 PRINT:"SCARSO";TAB(18);"MENO DI 150"
760 FOR T=1 TO 4000
770 NEXT T
780 CALL CLEAR
800 IF (C$><"A")+(C$><"O") THEN 830
810 PRINT TAB(3);"SENTI CAR";C$"   ";B$
820 GOTO 850
830 PRINT TAB(3);"VUOI GIOCARE ANCORA?"
840 GOTO 860
850 PRINT:TAB(2);"VUOI GIOCARE ANCORA?"::::::::
860 INPUT "(SI/NO)":A$
870 IF A$="SI" THEN 200
880 CALL CLEAR
890 PRINT "ALLORA CI VEDIAMO UN'ALTRA"
900 PRINT:"VOLTA E SPERO CHE AVRAI PIÙ"
910 PRINT:"FORTUNA!"
920 END
```

Spendiamo ora alcune parole per descrivere il funzionamento del gioco. Si tratta di riempire, muovendo il cursore (cioè il serpente) con gli appositi tasti «S»,«X»,«D»,«E» tutto il rettangolo all'interno del quale esso si trova. È vietato urtare uno dei bordi o la scia che via via il «serpente» lascia al suo passare. Quando ciò accade, il gioco si interrompe e viene visualizzato il punteggio raggiunto dal giocatore. Non vi raccontiamo come è stata ottenuta la formula che da il punteggio. Vi diciamo solo che essa tiene conto sia della velocità con cui si riempie lo schermo che della quantità di schermo che viene occupata (dalla scia del serpente).

Osserviamo che nelle linee 830 e 850 viene ripetuta una stessa frase. Si potrebbe pensare allora di metterla in una variabile a stringa e richiamarla quindi quando serve risparmiando così memoria.

È possibile fare anche altre migliorie e soprattutto abbellimenti: ma a ciò penserete voi!

11.4 **Un programma musicale: Per Elisa**

Abbiamo pensato di concludere questo capitolo sui giochi illustrando un programma (che non si può definire proprio un gioco) che fa uso delle possibilità sonore del TI 99/4A.

Si tratta innanzitutto di un programma volutamente non ottimizzato per dare la possibilità a coloro che, conoscendo la musica, vogliono confrontare lo spartito musicale con il listato.

Abbiamo inoltre pensato di non dotarlo di diagramma a blocchi (ossia il diagramma di flusso) in quanto ci è sembrato del tutto inutile. Si tratta infatti di una ripetizione della stessa sequenza di istruzioni; una tale sequenza è mostrata qui:

```
100 FOR I=1 TO 4
110 READ NOTA(I), VOLUME(I)
120 CALL SOUND(DCR,NOTA(I),VOLUME(I))
130 NEXT I
140 DATA........
```

Consigliamo l'utente di cercare di ottimizzare questo programma (magari con l'uso di più subroutine), dopo naturalmente averlo studiato, ed eventualmente documentarlo con qualche enunciato REM.

```
100 CALL SCREEN (16)
110 CALL CLEAR
120 PRINT TAB(10;"PER ELISA"
130 PRINT: TAB(3);"DI LUDWIG VAN BEETHOVEN"::::::::
140 FOR T=1 TO 1000
150 NEXT T
160 CALL SCREEN(1)
170 DSM=300
180 DCR=DSM/2
190 REM
200 REM
210 REM DSM: DURATA SEMIMINIMA
215 REM DCR: DURATA CROMA
220 REM
230 REM
240 GOSUB 260
250 GOTO 920
```

```
260 RESTORE
270 FOR I=1 TO 8
280 READ NOTA(I),VOLUME(I)
290 CALL SOUND(DCR,NOTA(I),VOLUME(I))
300 DATA 659,2,622,2,659,2,622,2,659,2,494,2,587,2,523,2
310 NEXT I
320 FOR I=1 TO 2
330 READ NOTA(I), VOLUME (I)
340 CALL SOUND(DCR,440,2,NOTA(I),VOLUME(I))
350 DATA 110,2,165,2,220,2
360 NEXT I
370 CALL SCREEN (16)
380 FOR I=1 TO 4
390 READ NOTA(I),VOLUME(I)
400 CALL SOUND(DCR,NOTA(I)VOLUME(I))
410 DATA 262,2,330,2,440,2
420 NEXT I
430 FOR I=1 TO 2
440 READ NOTA(I),VOLUME(I)
450 CALL SOUND(DCR,494,2,NOTA(I),VOLUME(I))
460 DATA 165,2,208,2
470 NEXT I
480 FOR I=1 TO 4
490 READ NOTA(I),VOLUME(I)
500 CALL SOUND(DCR,NOTA(I),VOLUME(I))
510 DATA 247,2,330,2,415,2,494,2
520 NEXT I
530 FOR I=1 TO 2
540 READ NOTA(I),VOLUME(I)
550 CALL SOUND(DCR,262,2,NOTA(I),VOLUME(I))
560 DATA 110,2,165,2
570 NEXT I
580 FOR I=1 TO 10
590 READ NOTA(I),VOLUME(I)
600 CALL SOUND(DCR,NOTA(I),VOLUME(I))
610 DATA 220,2,330,2,659,2,622,2,659,2,494,2,587,2,262,2
620 NEXT I
630 FOR I=1 TO 2
640 READ NOTA(I),VOLUME(I)
650 CALL SOUND(DCR,440,2,NOTA(I),VOLUME(I))
660 DATA 110,2,165,2
670 NEXT I
680 FOR I=1 TO 4
```

```
690 READ NOTA(I),VOLUME(I)
700 CALL SOUND(DCR,NOTA(I),VOLUME(I))
710 DATA 220,2,262,2,330,2,440,2
720 NEXT I
730 FOR I=1 TO 2
740 READ NOTA(I),VOLUME(I)
750 CALL SOUND(DCR,494,2,NOTA(I),VOLUME(I))
760 DATA 165,2,208,2
770 NEXT I
780 FOR I=1 TO 4
790 READ NOTA(I),VOLUME(I)
800 CALL SOUND(DCR,NOTA(I),VOLUME(I))
810 DATA 247,2,330,2,523,2,494,2
820 NEXT I
830 FOR I=1 TO 2
840 READ NOTA(I),VOLUME(I)
850 CALL SOUND(DCR,440,2,NOTA(I),VOLUME(I))
860 DATA 110,2,165,2
870 NEXT I
880 CALL SOUND(DCR,220,2)
890 FOR T=1 TO 80
900 NEXT T
910 RETURN
920 GOSUB 260
930 FOR I=1 TO 3
940 READ NOTA(I),VOLUME(I)
950 CALL SOUND(DCR,NOTA(I),VOLUME(I))
960 DATA 494,2,523,2,587,2
970 NEXT I
980 FOR I=1 TO 3
990 READ NOTA(I),VOLUME(I)
1000 CALL SOUND(DCR,659,1,NOTA(I),VOLUME(I))
1010 DATA 110,3,196,3,523,3
1020 NEXT I
1030 FOR I=1 TO 3
1040 READ NOTA(I),VOLUME(I)
1050 CALL SOUND(DCR,NOTA(I),VOLUME(I))
1060 DATA 392,2,698,2,659,2
1070 NEXT I
1080 FOR I=1 TO 3
1090 READ NOTA(I),VOLUME(I)
1100 CALL SOUND(DCR,587,2,NOTA(I),VOLUME(I))
1110 DATA 196,2,247,2,294,2
```

```
1120 NEXT I
1130 FOR I=1 TO 3
1140 READ NOTA(I),VOLUME(I)
1150 CALL SOUND(DCR,NOTA(I),VOLUME(I))
1160 DATA 349,2,659,2,587,2
1170 NEXT I
1180 FOR I=1 TO 3
1190 READ NOTA(I)
1200 CALL SOUND(DCR,523,2,NOTA(I),VOLUME(I))
1210 DATA 110,2,165,2,220,2
1220 NEXT I
1230 FOR I=1 TO 3
1240 READ NOTA(I),VOLUME(I)
1250 CALL SOUND(DCR,NOTA(I),VOLUME(I))
1260 DATA 330,2,587,2,523,2
1270 NEXT I
1280 FOR I=1 TO 2
1290 READ NOTA(I),VOLUME(I)
1300 CALL SOUND(DCR,494,2,NOTA(I),VOLUME(I))
1310 DATA 165,2,165,2
1320 NEXT I
1330 FOR I=1 TO 7
1340 READ NOTA(I),VOLUME(I)
1350 CALL SOUND(DCR,NOTA(I),VOLUME(I))
1360 DATA 330,2,330,2,659,2,330,2,659,2,659,2,1319,2
1370 NEXT I
1380 FOR I=1 TO 7
1390 READ NOTA(I),VOLUME(I)
1400 CALL SOUND(DCR,NOTA(I),VOLUME(I))
1410 DATA 622,2,659,2,622,2,659,2,622,2,659,2,622,2
1420 NEXT I
1430 GOSUB 260
1440 END
```

CONCLUSIONI

Voler concludere questa trattazione con poche parole non è una cosa semplice. Ci sarebbero tante raccomandazioni da fare ma, poiché noi riteniamo che questo libro sarà letto da persone veramente interessate all'apprendimento del BASIC e alle tecniche di programmazione, esse sarebbero per lo più superflue.

Ci limitiamo, quindi, a specificare alcuni punti che riteniamo essenziali.

Vi sarete resi conto che alcuni argomenti sono risultati un po' «nebulosi» alla prima lettura, cioè non se ne afferra esattamente il senso. Questa è una cosa che non desta stupore a chi sa che un qualsiasi testo va sempre riletto almeno due volte per essere compreso in pieno.

Consigliamo quindi sempre una seconda rilettura, soprattutto per argomenti come «La teoria degli Errori», che a primo acchitto lasciano perplessi; ma che appariranno, in seguito, utili e chiari.

C'è da specificare una cosa anche riguardo agli esercizi. Alcuni di essi (soprattutto quelli dei primi capitoli) possono apparire troppo teorici o un'inutile ripetizione di ciò che si è appena letto, ma noi abbiamo sperimentato che didatticamente sono molto utili soprattutto per fissare le idee.

Consigliamo quindi di non trascurarne alcuno e di provare sul calcolatore tutti quelli che ne consentono l'uso corredandoli, in seguito, di diagrammi di flusso.

A questo punto ringraziamo tutti coloro che hanno voluto dedicarci la loro attenzione ed auguriamo loro un buon lavoro.

GLOSSARIO DEI TERMINI TECNICI

Algoritmo	procedimento per la risoluzione di un problema.
Argomento	viene comunemente usato per indicare il valore associato ad un istruzione.
Array	collezione di elementi (numerici o alfanumerici).
Assembler	linguaggio la cui struttura è simile al linguaggio macchina; è composto da simboli mnemonici con argomenti esadecimali.
Basic	linguaggio simbolico usato in genere nei personal ed home-computer.
Sistema binario	sistema di numerazione il cui alfabeto è costituito da due simboli (0 e 1).
Bit	unità elementare di informazione. Può assumere solo i valori 0 ed 1.
Byte	insieme di 8 bit; costituisce il più piccolo elemento di meoria indirizzabile.
Capacità	quantità di informazioni che una memoria può contenere. In genere si misura in k-byte.
Carattere	un numero, una lettera, un segno di punteggiatura o uno speciale simbolo grafico.
Ciclo	vedi definizione di loop.
Codice ASCII	è un codice usato in moltissimi calcolatori per rappresentare i caratteri con dei numeri (interi e compresi tra 0 e 128).
Concatenazione :	unione di due o più stringhe effettuato tramite il simbolo & (e commerciale).
Costante	valore numerico o alfanumerico noto.

C.P.U.	(Central Processor Unit) unità centrale di elaborazione. È l'unità che prende le decisioni, svolge le operazioni e le funzioni di controllo.
Cursore	elemento intermittente che indica sul video la posizione in cui verrà inserito il prossimo carattere.
Dato	con questo termine si indica qualsiasi tipo di informazione (numerica o alfanumerica) che l'utente da al calcolatore o viceversa.
Debugging	termine usato per indicare la ricerca degli errori.
Default	valore assegnato automaticamente dal programma all'argomento di alcune funzioni, quando l'utente non lo specifica altrimenti.
Diagramma di flusso	rappresentazione grafica di un algoritmo.
Disco	supporto di natura magnetica attraverso il quale è possibile registrare e mantenere nel tempo programmi e/o file di dati.
Sistema esadecimale	sistema di numerazione in base 16 i cui simboli sono: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.
Esecuzione	atto di obbedire, da parte del calcolatore, alle istruzioni contenute in un programma.
Espressione	insieme di costanti e variabili correlate da operatori. Può essere numerica o alfanumerica.
File	collezione di records di dati memorizzati su nastro o disco magnetico.
Floppy disk drive:	dispositivo periferico usato per memorizzare programmi e dati su dischi di plastica flessibile ricoperti di materiale magnetico (spesso chiamati floppy-disk o diskette).
Flow-chart	termine inglese che indica diagramma di flusso.
Funzione	definizione di una procedura matematica (che opera su argomenti numerici o alfanumerici).

Hardware — con questo termine si indica tutto ciò che riguarda l'aspetto fisico del calcolatore e delle apparecchiature ad esso collegate: tastiera, video, unità disco, stampante, ecc.

Incremento — valore che periodicamente modifica una variabile (ad es. l'argomento dello STEP).

Indirizzo — numero che identifica univocamente la posizione di una cella di memoria.

Interprete — programma che serve a tradurre in linguaggio macchina un linguaggio simbolico durante l'esecuzione del programma stesso.

kbyte — multiplo del byte. 1kbyte = 1024 byte.

Linguaggio macchina — linguaggio numerico direttamente comprensibile dal calcolatore e composto solo da cifre binarie (0 e 1).

Listato — stampa delle linee di istruzioni di cui è formato un programma. Tale stampa può essere eseguita sia su monitor che su stampante.

Locazione — particolare posizione in memoria individuata da un suo indirizzo.

Loop — è un insieme di linee di programma consecutive ripetuto un numero definito o indefinito di volte.

Mantissa — base numerica di un numero espresso in notazione scientifica.

Matrice — struttura di dati costituita da un insieme finito di elementi; sono organizzati secondo righe e colonne individuate da indici.

Memoria — parte del calcolatore che contiene i dati ed i programmi. Si divide in memoria centrale e memoria di massa.

Memoria centrale — è quella parte di memoria dove vengono registrati temporaneamente dati e istruzioni relative a programmi dell'utente (RAM). In più, in essa, è contenuto anche il sistema operativo e l'interprete Basic (ROM).

Memoria di massa	memoria ausiliaria costituita da tutti i dispositivi esterni (unità a dischi, registratori a cassette, film fotografici, ecc.) che servono per immagazzinare dati e programmi che il calcolatore userà a richiesta dell'utente.
Microcalcolatore :	calcolatore la cui unità centrale è un microprocessore.
Microprocessore:	CPU realizzata su singolo circuito integrato.
Numero pseudo-casuale	numero casuale prodotto con una ben determinata procedura.
Operatore	simbolo usato per legare variabili costanti o espressioni. Può essere di tre tipi: relazionale, aritmetico o di concatenazione.
Parametro	argomento di una funzione.
Periferiche	con questo nome si indicano tutti i dispositivi di I/O (ingresso/uscita) quali: tastiera, monitor; stampante, perforatrice di scheda, ecc.
Programma	sequenza ordinata di istruzioni che il calcolatore esegue sequenzialmente ed ordinatamente.
Programmazione : strutturata	metodo di programmazione TOP-DOWN.
Prompt	è un simbolo posto prima del cursore e che segnala che il calcolatore è pronto per ricevere comandi in genere.
RAM	(Random Acces Memory) memoria che contiene i programma dell'utente.
ROM	(Read Only Memory) memoria a sola lettura. Nei calcolatori basati su microprocessore è il componente su cui è memorizzato, in modo permanente, il programma.
Record	collezione di dati correlati tra di loro.
Salto	procedura che modifica la normale esecuzione delle istruzioni.

Salto condizionato	avviene quando il controllo è trasferito in base al risultato di alcune operazioni aritmetiche o logiche.
Salto incondizionato	avviene se si trasferisce il controllo del programma incondizionatamente (istruzione GOTO del TI-BASIC).
Scrolling	atto svolto dal computer quando muove informazioni presenti sul video (generalmente verso l'alto) per lasciare spazio ad altre in arrivo.
Sistema operativo	insieme di programmi residenti permanentemente nella macchina e che gestiscono le risorse del calcolatore.
Software	è l'insieme di tutti i programmi che sono presenti o che possono essere inseriti nel calcolatore. Si distingue in Software di base: tutti i programmi contenuti nelle ROM (sistema operativo, interprete Basic, ecc.), e Software applicativo (ossia tutti i programmi fatti dall'utente).
Sottoprogramma : (subroutine)	parte di programma progettata per risolvere determinate procedure che ricorrono spesso in una elaborazione.
Stringa :	sequenza di lettere, numeri o simboli e trattati come una singola unità.
TOP-DOWN	metodo di approccio alla risoluzione di un problema: esso consiste nel fare vari schemi di risoluzione via via più dettagliati (dal generale al particolare).
Unità di Input	unità di ingresso che fornisce informazioni (dati, programmi, ecc) al calcolatore: tastiera, video, lettore di schede, ecc.
Unità di Output :	unità di uscita che riceve dal calcolatore informazioni (dati, risultati in genere, ecc.); esempi di ciò sono: video, stampante, altri calcolatori, ecc.
Utente	colui che utilizza il calcolatore.
Variabile	cella di memoria a cui si da un nome e si assegna un contenuto che viene letto richiamando semplicemente il nome della variabile.
Vettore :	lista ordinata di elementi ognuno dei quali è individuato da un indice.

BIBLIOGRAFIA

La consultazione necessaria per lo sviluppo del suddetto volume si è basata sull'uso dei seguenti testi.

1. Gottfried B.S. *Programmare in BASIC* - Mc Graw-Hill, 1975

2. Andronico A. et al. *Scienza degli elaboratori* - Zanichelli, 1971

3. Lipschutz A. Poe *Programmare in FORTRAN* - McGraw Hill, 1975

4. Siciliano A. *Il linguaggio FORTRAN* - Zanichelli, 1974

5. Lamoitier J.P. *50 esercizi in BASIC* - G.E. Jackson, 1982

6. Verri G. *Calcolatori elettronici e linguaggio FORTRAN* - V. Veschi, 1979

7. Haut H. *Programmi di matematica e statistica in BASIC* - G.E. Jackson, 1982

8. Townsend R. *Learning to use the TI 99/4A Computer* - Gower, 1983

9. Peckham H.D. *Programming BASIC with the Ti Home Computer* - Mc Graw Hill, 1979

10. Le Beux P. *Introduzione al BASIC* - G.E. Jackson, 1980

11. *Norma AFNOR Z67-00 Flow-chart* - Aprile 1966

12. *Manuale d'uso del TI 99/4A.*

M. PEROTTI

Impariamo a programmare in TI BASIC

G. CANGEMI